AdaTag: Multi-Attribute Value Extraction from Product Profiles with Adaptive Decoding

Jun Yan\(^1\)*, Nasser Zalmout\(^2\), Yan Liang\(^2\), Christian Grant\(^3\),
Xiang Ren\(^1\), Xin Luna Dong\(^2\)
University of Southern California\(^1\), Amazon.com\(^2\), University of Oklahoma\(^3\),
\{yanjun, xiangren\}@usc.edu, \{nzalmout, ynliang, lunadong\}@amazon.com, cgrant@ou.edu

Abstract

Automatic extraction of product attribute values is an important enabling technology in e-Commerce platforms. This task is usually modeled using sequence labeling architectures, with several extensions to handle multi-attribute extraction. One line of previous work constructs attribute-specific models, through separate decoders or entirely separate models. However, this approach constrains knowledge sharing across different attributes. Other contributions use a single multi-attribute model, with different techniques to embed attribute information. But sharing the entire network parameters across all attributes can limit the model’s capacity to capture attribute-specific characteristics. In this paper we present AdaTag, which uses adaptive decoding to handle extraction. We parameterize the decoder with pretrained attribute embeddings, through a hypernetwork and a Mixture-of-Experts (MoE) module. This allows for separate, but semantically correlated, decoders to be generated on the fly for different attributes. This approach facilitates knowledge sharing, while maintaining the specificity of each attribute. Our experiments on a real-world e-Commerce dataset show marked improvements over previous methods.

1 Introduction

The product profiles on e-Commerce platforms are usually comprised of natural texts describing products and their main features. Key product features are conveyed in unstructured texts, with limited impact on machine-actionable applications, like search (Ai et al., 2017), recommendation (Kula, 2015), and question answering (Kulkarni et al., 2019), among others. Automatic attribute value extraction aims to obtain structured product features from product profiles. The input is a textual sequence from the product profile, along with the required attribute to be extracted, out of potentially large number of attributes. The output is the corresponding extracted attribute values. Figure 1 shows the profile of a moisturizing cream product as an example, which consists of a title, several information bullets, and a product description. It also shows the attribute values that could be extracted.

Most existing studies on attribute value extraction use neural sequence labeling architectures (Zheng et al., 2018; Karamanolakis et al., 2020; Xu et al., 2019). To handle multiple attributes, one line of previous contributions develops a set of “attribute-specific” models (i.e., one model per attribute). The goal is to construct neural networks with (partially) separate model parameters for different attributes. For example, one can construct an independent sequence labeling model for each attribute and make predictions with all the models collectively (e.g., the vanilla OpenTag model (Zheng et al., 2018)). Instead of totally separate models, one can also use different tag sets corresponding to different attributes. These networks can also share the feature encoder and use separate

---

* Most of the work was done during an internship at Amazon.
label decoders (Yang et al., 2017). However, the explicit network (component) separation in these contributions constrains knowledge-sharing across different attributes. Exposure to other attributes can help in disambiguating the values for each attribute. And having access to the entire training data for all attributes helps with the generic sequence tagging task. Another line for multi-attribute extraction contributions learns a single model for all attributes. The model proposed by Xu et al. (2019), for example, embeds the attribute name with the textual sequence, to achieve a single “attribute-aware” extraction model for all attributes. This approach addresses the issues in the previous direction. However, sharing all the network parameters with all attributes could limit the model’s capacity to capture attribute-specific characteristics.

In this paper we address the limitations of the existing contribution lines, through adaptive decoder parameterization. We propose to generate a decoder for each attribute based on its embedding. This results in different but semantically correlated decoders, which maintain the specific characteristics for each attribute, while facilitating knowledge-sharing across different attributes. To this end, we use conditional random fields (CRF) (Lafferty et al., 2001) as the decoders, and parameterize the decoding layers with the attribute embedding through a hypernetwork (Ha et al., 2017) and a Mixture-of-Experts (MoE) module (Jacobs et al., 1991). We further explore several pretrained attribute embedding techniques, to add useful attribute-specific external signals. We use both contextualized and static embeddings for the attribute name along with its potential values to capture meaningful semantic representations.

We summarize our contributions as follows: (1) We propose a multi-attribute value extraction model with an adaptive CRF-based decoder. Our model allows for knowledge sharing across different attributes, yet maintains the individual characteristics of each attribute. (2) We propose several attribute embedding methods, that provide important external semantic signals to the model. (3) We conduct extensive experiments on a real-world e-Commerce dataset, and show improvements over previous methods. We also draw insights on the behavior of the model and the attribute value extraction task itself.

2 Background

2.1 Problem Definition

The main goal of the task is to extract the corresponding values for a given attribute, out of a number of attributes of interest, from the text sequence of a product profile. Formally, given a text sequence $X = [x_1, \ldots, x_n]$ in a product profile, where $n$ is the number of words, and a query attribute $r \in R$, where $R$ is a predefined set of attributes, the model is expected to extract all text spans from $X$ that could be valid values for attribute $r$ characterizing this product. When there are no corresponding values mentioned in $X$, the model should return an empty set. For example, for the product in Figure 1, given its title as $X$, the model is expected to return (“Dry”, “Sensitive”) if $r$ = “SkinType”, and an empty set if $r$ = “Color”.

Following standard approaches (Zheng et al., 2018; Xu et al., 2019; Karamanolakis et al., 2020), under the assumption that different values for an attribute do not overlap in the text sequence, we formulate the value extraction task as a sequence tagging task with the BIOE tagging scheme. That is, given $X$ and $r$, we want to predict a tag sequence $Y = [y_1, \ldots, y_n]$, where $y_i \in \{B, I, O, E\}$ is the tag for $x_i$. “B”/“E” indicates the corresponding word is the beginning/ending of an attribute value, “I” means the word is inside an attribute value, and “O” means the word is outside any attribute value. Table 1 shows an example of the tag sequence for attribute “Scent” of a shower gel collection, where “orchid”, “cherry pie”, “mango ice cream” could be extracted as the values.

<table>
<thead>
<tr>
<th>$X$</th>
<th>orchid / cherry pie / mango ice cream scent</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Y$</td>
<td>B O B E O B I E O</td>
</tr>
</tbody>
</table>

Table 1: An example of the tag sequence for attribute “Scent” annotated with the BIOE scheme.

2.2 BiLSTM-CRF Architecture

The BiLSTM-CRF architecture (Huang et al., 2015) consists of a BiLSTM-based text encoder, and a CRF-based decoder. This architecture has been proven to be effective for the attribute value extraction task (Zheng et al., 2018; Xu et al., 2019; Karamanolakis et al., 2020). We build our AdaTag model based on the BiLSTM-CRF architecture as we find that the BiLSTM-CRF-based models generally perform better than their BiLSTM-based,
BERT-based (Devlin et al., 2019) and BERT-CRF-based counterparts, as shown in §5. We introduce the general attribute-agnostic BiLSTM-CRF architecture, which our model is based on, in this subsection.

Given a text sequence $X = [x_1, \ldots, x_n]$. We obtain the sequence of word embeddings $X = [x_1, \ldots, x_n]$ using an embedding matrix $W_{\text{word}}$. We get the hidden representation of each word by feeding $X$ into a bi-directional Long-Short Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) layer with hidden size $d_h$:

$$[h_1, \ldots, h_n] = \text{BiLSTM}([x_1, \ldots, x_n]).$$  (1)

We use a CRF-based decoder to decode the sequence of hidden representations while capturing the dependency among tags (e.g., “I” can only be followed by “E”). It consists of a linear layer and a transition matrix, which are used to calculate the emission score and the transition score for the tag prediction respectively. Let $V = [B, I, O, E]$ be the vocabulary of all possible tags. We calculate an emission score matrix $P = [p_1, \ldots, p_n] \in \mathbb{R}^{4 \times n}$, where $P_{ij}$ is the score for assigning the $i$-th tag in $V$ to $x_j$. This is computed by feeding $[h_1, \ldots, h_n]$ into a linear layer with parameters $[W, b]$, specifically $p_i = Wh_i + b \in \mathbb{R}^4$, where $W \in \mathbb{R}^{4 \times d_h}$ and $b \in \mathbb{R}^4$. For a BIOE tag sequence $Y = [y_1, \ldots, y_n]$, we get its index sequence $Z = [z_1, \ldots, z_n]$ where $z_i \in \{1, 2, 3, 4\}$ is the index of $y_i$ in $V$. The score for an input text sequence $X$ to be assigned with a tag sequence $Y$ is calculated as:

$$s(X, Y) = s(X, Z) = \sum_{i=1}^{n-1} T_{z_i, z_{i+1}} + \sum_{i=1}^{n} P_{z_i},$$  (2)

where $T \in \mathbb{R}^{4 \times 4}$ is the transition matrix of CRF, such that $T_{ij}$ is the score of a transition from the $i$-th tag to the $j$-th tag in $V$.

3 Method

3.1 Model Overview

The multi-attribute value extraction task can be thought of as a group of extraction subtasks, corresponding to different attributes. While all attributes share the general knowledge about value extraction, each has its specificity. The key idea in our proposed model is to dynamically adapt the parameters of the extraction model based on the specific subtask corresponding to the given attribute. We use a BiLSTM-CRF (Huang et al., 2015) architecture, where different subtasks, corresponding to different attributes, share the same text encoder to derive a contextualized hidden representation for each word. Then the hidden representations of the text sequence are decoded into a sequence of tags with a CRF-based decoder, the parameters of which are generated on the fly based on the attribute embedding. This facilitates a knowledge-sharing scheme across different attributes. Intuitively, this can help with learning generic abilities like detecting value boundary, which is at the core of the extraction process of any attribute. At the same time, our model provides each subtask with a customized decoder parameterization, which improves the model’s capacity for capturing attribute-specific knowledge. Figure 2 presents our overall model architecture, where we equip the BiLSTM-CRF architecture with an adaptive CRF-based decoder. In §3.2, we will introduce our adaptive CRF-based decoder which is parameterized with the attribute embedding. In §3.3, we will describe how to obtain pre-trained attribute embeddings that can capture the characteristics of different subtasks, so that “similar” attributes get “similar” decoding layers.

3.2 Adaptive CRF-based Decoder

In attribute value extraction, the model takes the text sequence $X$ with a query attribute $r$ as input, and is expected to predict $Y$ based on both $X$ and $r$. To make the model aware of the query attribute, we need to incorporate the attribute information into some components of the BiLSTM-CRF architecture. The BiLSTM-based text encoder is responsible for encoding the text sequence and obtain a contextualized representation for each word, which can be regarded as “understanding” the sentence. The CRF-based decoder then predicts a tag for each word based on its representation. Therefore, we propose that all attributes share a unified text encoder so that the representation can be enhanced through learning with different subtasks, and each attribute has a decoder adapted to its corresponding subtask, the parameters of which are generated based on the attribute information.

As introduced in §2.2, a CRF-based decoder consists of a linear layer and a transition matrix. The linear layer takes hidden representations as input,
and predicts a tag distribution for each word independently. It captures most of characteristics of value extraction for a given attribute based on the text understanding. More flexibility is needed to model the specificity of different attributes. By contrast, the transition matrix learns the dependency among tags to avoid predicting unlikely tag sequence. It only captures shallow characteristics for the attribute based on its value statistics. For example, the transition scores form “B” to other tags largely depend on the frequent lengths of the attribute values. If single-word values are mentioned more often, then “B” is more likely to be followed by “O”. If two-word values dominate the vocabulary, then “B” is more likely to be followed by “E”. Attributes could be simply clustered based on these shallow characteristics.

In this work we parameterize the CRF-based decoder with the attribute embedding \( r \in \mathbb{R}^{d_r} \), where \( d_r \) is the dimension of the attribute embedding. For the linear layer, we adopt a hypernetwork (Ha et al., 2017) due to its high flexibility. For the transition matrix, we develop a Mixture-of-Experts (Pahuja et al., 2019) module to leverage the latent clustering nature of attributes. We nevertheless experiment with all 4 combinations of these methods in §5.3, and this choice does the best.

**Hypernetwork.** The idea of hypernetworks (Ha et al., 2017) is to use one network to generate the parameters of another network. Such approach has high flexibility when no constraint is imposed during generation. We therefore use it to parameterize the linear layer. In our model, we learn two different linear transformations that map the attribute embedding to the parameters of the linear layer (\( W \in \mathbb{R}^{4 \times d_r}, b \in \mathbb{R}^4 \)) in the CRF-based decoder:

\[
W = \text{Reshape}(W_{\text{hyper}}^w + b_{\text{hyper}}^w), \quad b = \text{Reshape}(W_{\text{hyper}}^b + b_{\text{hyper}}^b). \tag{3}
\]

Here \( W_{\text{hyper}}^w \in \mathbb{R}^{4d_h \times d_r}, b_{\text{hyper}}^w \in \mathbb{R}^{4d_h}, W_{\text{hyper}}^b \in \mathbb{R}^{4 \times d_r}, b_{\text{hyper}}^b \in \mathbb{R}^4 \), and the \text{Reshape} operator reshapes a 1-D vector into a matrix with the same number of elements.

**Mixture-of-Experts.** The idea of Mixture-of-Experts (Jacobs et al., 1991) is to have a group of networks (“experts”) that jointly make decisions with dynamically determined weights. Unlike previous approaches that combine each expert’s prediction, we combine their parameters for generating the transition matrix. Let \( k \) be the number of experts we use to parameterize the transition matrix \( T \in \mathbb{R}^{4 \times 4} \) where \( k \) is a hyperparameter. We introduce \( k \) learnable matrices \( T^{(1)}, \ldots, T^{(k)} \) for the \( k \) experts. Each expert’s matrix can be understood as a cluster prototype and we employ a linear gating network to compute the probability of assigning the given attribute to each expert:

\[
\lambda = \text{Softmax}(W_{\text{moe}}^w r + b_{\text{moe}}^w). \quad \text{Here } W_{\text{moe}}^w \in \mathbb{R}^{k \times d_r}, b_{\text{moe}}^w \in \mathbb{R}^k, \lambda = [\lambda_1, \ldots, \lambda_k] \in \mathbb{R}^k \quad \text{and } \sum_{i=1}^{k} \lambda_i = 1.
\]

The parameters for the transition matrix for this attribute is calculated as:

\[
T = \sum_{i=1}^{k} \lambda_i T^{(i)}.
\]

### 3.3 Pretrained Attribute Embeddings

The attribute embedding \( r \) plays a key role in deriving the attribute-specific decoding layers. Therefore, the quality of the attribute embeddings is crucial to the success of our parameterization method. Good attribute embeddings are supposed to capture the subtask similarities such that similar extraction tasks use decoders with similar parameters. In this work, we propose to use the attribute name and possible values as a proxy to capture the characteristics of the value extraction task for a given attribute. The attribute embeddings can therefore be directly derived from the training data and loaded into the attribute embedding layer as initialization.

For each attribute \( r \), we first collect all the sen-
tences from the training data that are annotated with at least one value for \( r \). We denote the collected sentences with values as \( D_r = \{(\tilde{r}, v_i, X_i)\}_{i=1}^{n_r} \)
where \( \tilde{r} \) is the phrase representation of \( r \) (e.g., \( \tilde{r} = “\text{Skin Type}“ \) if \( r = “\text{SkinType}“ \). \( v_i \) is a span in text sequence \( X_i \) that serves as the value for \( r \), and \( n_r \) is the number of collected sentences. For each \((\tilde{r}, v_i, X_i)\), we can calculate an attribute name embedding \( \mathbf{r}_i^{\text{name}} \) and an attribute value embedding \( \mathbf{r}_i^{\text{value}} \) in either a contextualized way or an uncontextualized way, which are detailed later. We pool over all instances in \( D_r \) to get the final attribute name embedding and attribute value embedding, which are concatenated as the attribute embedding:

\[
\mathbf{r}_i = \frac{1}{n_r} \sum_{i=1}^{n_r} \mathbf{r}_i^{\text{name}}, \mathbf{r}_i^{\text{value}} = \frac{1}{n_r} \sum_{i=1}^{n_r} \mathbf{r}_i^{\text{value}}, \\
r = \text{Concat}(\mathbf{r}_i^{\text{name}}, \mathbf{r}_i^{\text{value}}).
\]

**Contextualized Embeddings.** Taking the context into consideration helps get embeddings that can more accurately represent the semantics of the word. Here we use the contextualized representations provided by BERT (Devlin et al., 2019) to generate the embedding. We use BERT to encode \( X_i \) and get \( v_i \)’s phrase embedding (the averaged embedding of each word in the phrase) as \( \mathbf{r}_i^{\text{value}} \). By replacing \( v_i \) with “[BOA] \( \tilde{r} \) [EOA]”\(^1\) and encoding the modified sequence with BERT, we get the phrase embedding for “[BOA] \( \tilde{r} \) [EOA]” as \( \mathbf{r}_i^{\text{name}} \).

**Uncontextualized Embeddings.** Static embeddings like Word2Vec (Mikolov et al., 2013) and Glove (Pennington et al., 2014) can be more stable to use under noisy contexts. We use Glove (50d) to get the phrase embedding for \( v_i \) as \( \mathbf{r}_i^{\text{value}} \) and the phrase embedding for \( \tilde{r} \) as \( \mathbf{r}_i^{\text{name}} \).

### 3.4 Model Training

As we parameterize the CRF-based decoder with the attribute embedding through MoE and hypernetwork, the learnable parameters in our model includes \( \theta_{\text{encoder}} = \{\mathbf{W}_\text{word}, \theta_{\text{bi-lstm}}\} \), \( \theta_{\text{hyper}} = \{\mathbf{W}_\text{hyper}^b, \mathbf{b}_\text{hyper}^b, \mathbf{W}_\text{hyper}^t, \mathbf{b}_\text{hyper}^t\} \), \( \theta_{\text{moe}} = \{\mathbf{W}_\text{moe}, \mathbf{b}_\text{moe}, \{\mathbf{T}(i)\}_{i=1}^{k}\} \). We freeze the attribute embeddings \( \mathbf{W}_\text{att} \) as it gives better performance, which is also discussed in §5.3.

The whole model is trained end-to-end by maximizing the log likelihood of \( (X, r, Y) \) triplets in the training set, which is derived from Equation 2

\[
s(X, r, Y) = \sum_{i=0}^{n} T_{z_{i+1}} + \sum_{i=1}^{n} P_{z_i},
\]

\[
\log p(Y | X, r) = \log \frac{s(X, r, Y)}{\sum_{Y' \in V} s(X, r, Y')},
\]

where \( V_n \) is the set of all tag sequences of length \( n \). The log likelihood can be computed efficiently using the forward algorithm (Baum and Eagon, 1967) for hidden Markov models (HMMs). At inference, we adopt Viterbi algorithm (Viterbi, 1967) to get the most likely \( Y \) given \( X \) and \( r \) in test set.

### 4 Experimental Setup

#### 4.1 Dataset

To evaluate the effectiveness of our proposed model, we build a dataset by collecting product profiles (title, bullets, and description) from the public web pages at Amazon.com.\(^2\)

Following previous works (Zheng et al., 2018; Karamanolakis et al., 2020; Xu et al., 2019), we obtain the attribute-value pairs for each product using the product information on the webpages by distant supervision. We select 32 attributes with different frequencies. For each attribute, we collect product profiles that are labeled with at least one value for this attribute. We further split the collected data into training (90%) and development (10%) sets.

The annotations obtained by distant supervision are often noisy so they cannot be considered as gold labels. To ensure the reliability of the evaluation results, we also manually annotated an additional testing set covering several attributes. We randomly selected 12 attributes from the 32 training attributes, took a random sample from the relevant product profiles for each attribute, and asked human annotators to annotate the corresponding values. We ensured that there is no product overlapping between training/development sets and the test set.

Putting together the datasets built for each individual attribute, we end up with training and development sets for 32 attributes, covering 333,857 and 40,008 products respectively. The test set has 12 attributes and covers 11,818 products. Table 2 presents the statistics of our collected dataset. Table 3 shows the attribute distribution of the training

\(^1\) [BOA] and [EOA] are special tokens that are used to separate the attribute name from context in the synthetic sentence.

\(^2\) While Xu et al. (2019) released a subset of their collected data from AliExpress.com, their data has a long-tailed attribute distribution (7650 of 8906 attributes occur less than 10 times). It brings major challenges for zero-shot learning, which are beyond our scope.
set. It clearly demonstrates the data imbalance issue of the real-world attribute value extraction data.

Most of the attribute values are usually covered in the title and bullets, since sellers would aim to highlight the product features early on in the product profile. The description, on the other hand, can provide few new values complementing those mentioned in the title and bullets, but significantly increases the computational costs due to its length. Therefore, we consider two settings for experiments: extracting from the title only (“Title”) and extracting from the concatenation of the title and bullets (“Title + Bullets”).

<table>
<thead>
<tr>
<th>Split</th>
<th># Attributes</th>
<th># Products</th>
<th>Avg. # Words (Title)</th>
<th>Avg. # Words (Title+Bullets)</th>
</tr>
</thead>
<tbody>
<tr>
<td>train</td>
<td>32</td>
<td>333,857</td>
<td>20.9</td>
<td>113.4</td>
</tr>
<tr>
<td>dev</td>
<td>32</td>
<td>40,008</td>
<td>21.0</td>
<td>113.7</td>
</tr>
<tr>
<td>test</td>
<td>12</td>
<td>11,818</td>
<td>20.5</td>
<td>120.0</td>
</tr>
</tbody>
</table>

Table 2: Statistics of our collected dataset.

<table>
<thead>
<tr>
<th># Products</th>
<th># Attr.</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>[10000, 50279]</td>
<td>12</td>
<td>Color, Flavor, SkinType, HairType</td>
</tr>
<tr>
<td>[1000, 10000]</td>
<td>10</td>
<td>ActiveIngredients, CaffeineContent</td>
</tr>
<tr>
<td>[100, 1000]</td>
<td>6</td>
<td>SpecialIngredients, DosageForm</td>
</tr>
<tr>
<td>[15, 100]</td>
<td>4</td>
<td>PatternType, ItemShape</td>
</tr>
</tbody>
</table>

Table 3: Frequencies of different attributes in the training set.

4.2 Evaluation Metrics

For each attribute, we calculate Precision/Recall/F₁ based on exact string matching. That is, an extracted value is considered correct only if it exactly matches one of the ground truth values for the query attribute in the given text sequence. We use Macro-Precision/Macro-Recall/Macro-F₁ (denoted as P/R/F₁) as the aggregated metrics to avoid bias towards high-resource attributes. They are calculated by averaging per-attribute metrics.

4.3 Compared Methods

We compare our proposed model with a series of strong baselines for attribute value extraction.³

**BiLSTM** uses a BiLSTM-based encoder. Each hidden representation is decoded independently into a tag with a linear layer followed by softmax. **BiLSTM-CRF** (Huang et al., 2015) uses a BiLSTM-based encoder and a CRF-based decoder, as described in §2.2. Zheng et al. (2018) propose OpenTag, which uses a self-attention layer between the BiLSTM-based encoder and CRF-based decoder for interpretable attribute value extraction. However, we find the self-attention layer not helpful for the performance.⁴ We therefore only present the results for BiLSTM-CRF in §5. **BERT** (Devlin et al., 2019) and **BERT-CRF** replace the BiLSTM-based text encoder with BERT.⁵

Note that these four methods don’t take the query attribute as input. To make them work in our more realistic setting with multiple (N) attributes, we consider two variants for each of them. (1) “N tag sets”: We introduce one set of B/I/E tags for each attribute, so that a tag sequence can be unambiguously mapped to the extraction results for multiple attributes. For example, the tag sequence “B-SkinType E-SkinType O B-Scent” indicates that the first two words constitutes a value for attribute SkinType, and the last word is a value for Scent. Only one model is needed to handle the extraction for all attributes. (2) “N models”: We build one value extraction model for each attribute — we’ll train N models for this task.

The “N models” variant isolates the learning of different attributes. To enable knowledge sharing, other methods share the model components or the whole model among all attributes: **BiLSTM-CRF-SharedEmb** shares a word embedding layer among all attributes. Each attribute has its own BiLSTM layer and CRF-based decoder, which are independent from each other. **BiLSTM-MultiCRF** (Yang et al., 2017) shares a BiLSTM-based text encoder among all attributes. Each attribute has its own CRF-based decoder. **SUOpenTag** (Xu et al., 2019) encodes both the text sequence and the query attribute with BERT and adopts a cross-attention mechanism to get an attribute-aware representation for each word. The hidden representations are decoded into tags with a CRF-based decoder.

We also include **AdaTag** (Random AttEmb), which has the same architecture as our model but uses randomly initialized learnable attribute embeddings of the same dimension.

4.4 Implementation Details

We implement all models with PyTorch (Paszke et al., 2019). For models involving BERT, we use the bert-base-cased version. Other models use pretrained 50d Glove (Pennington et al., 2014)⁴

³We discuss the sizes of different models in Appendix §A.
⁴We hypothesize that the improvement brought by the self-attention module is dataset-specific.
⁵The hidden representation for each word is the average of its subword representations.
Table 4: Performance comparison on test set with 12 attributes (best in boldface and second best underlined).

<table>
<thead>
<tr>
<th>Methods</th>
<th>Title + Bullets</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P(%) R(%) F₁(%)</td>
<td>P(%) R(%) F₁(%)</td>
</tr>
<tr>
<td>Group I: N tag sets</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BiLSTM (N tag sets)</td>
<td>35.15 54.28 38.92</td>
<td>32.17 34.30 31.18</td>
</tr>
<tr>
<td>BiLSTM-CRF (N tag sets)</td>
<td>35.23 53.94 38.85</td>
<td>34.03 35.01 32.11</td>
</tr>
<tr>
<td>BERT (N tag sets)</td>
<td>33.52 50.48 36.29</td>
<td>31.41 30.62 28.26</td>
</tr>
<tr>
<td>BERT-CRF (N tag sets)</td>
<td>34.55 51.96 37.45</td>
<td>32.63 31.24 28.89</td>
</tr>
<tr>
<td>Group II: N models</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BiLSTM (N models)</td>
<td>64.37 71.71 64.64</td>
<td>61.61 60.26 58.56</td>
</tr>
<tr>
<td>BiLSTM-CRF (N models)</td>
<td>63.94 72.14 64.78</td>
<td>62.07 61.46 59.19</td>
</tr>
<tr>
<td>BERT (N models)</td>
<td>55.34 72.86 58.48</td>
<td>53.35 61.27 54.37</td>
</tr>
<tr>
<td>BERT-CRF (N models)</td>
<td>54.29 72.79 57.49</td>
<td>49.25 59.33 50.49</td>
</tr>
<tr>
<td>Group III: shared components</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BiLSTM-CRF-SharedEmb</td>
<td>63.77 72.50 64.62</td>
<td>58.95 60.58 57.66</td>
</tr>
<tr>
<td>BiLSTM (N models)</td>
<td>64.48 72.04 64.81</td>
<td>60.64 62.75 59.78</td>
</tr>
<tr>
<td>BERT (N models)</td>
<td>63.62 71.67 64.76</td>
<td>61.57 60.48 59.62</td>
</tr>
<tr>
<td>BERT-CRF (N models)</td>
<td>64.80 71.95 65.74</td>
<td>60.14 62.14 60.04</td>
</tr>
<tr>
<td>AdaTag (Random AttEmb)</td>
<td>65.00 75.87 67.48</td>
<td>62.87 62.45 60.87</td>
</tr>
</tbody>
</table>

Table 5: Performance comparison on high-resource and low-resource attributes.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P(%) R(%) F₁(%)</td>
<td>P(%) R(%) F₁(%)</td>
</tr>
<tr>
<td>BiLSTM-CRF (N models)</td>
<td>54.94 75.66 61.57</td>
<td>83.72 65.08 71.19</td>
</tr>
<tr>
<td>BiLSTM-MultiCRF</td>
<td>54.38 74.42 60.23</td>
<td>84.70 67.29 73.97</td>
</tr>
<tr>
<td>SUOpenTag</td>
<td>55.34 72.94 60.49</td>
<td>80.16 69.13 73.31</td>
</tr>
<tr>
<td>AdaTag (Our Model)</td>
<td>56.05 76.07 62.00</td>
<td>82.90 75.48 78.45</td>
</tr>
</tbody>
</table>

5 Experimental Results

5.1 Overall Results

Table 4 presents the overall results using our dataset under both “Title” and “Title + Bullets” settings. Our model demonstrates great improvements over baselines on all metrics except getting second best recall under the “Title + Bullets” settings. The comparisons demonstrate the overall effectiveness of our model and pretrained attribute embeddings.

The “N tag sets” variants get much lower performance than other methods, probably due to the severe data imbalance issue in the training set (see Table 3). All attributes share the same CRF-based decoder, which could make learning biased towards high-resource attributes. Note that introducing one set of tags for each entity type is the standard approach for the Named Entity Recognition (NER) task. Its low performance suggests that the attribute value extraction task is inherently different from standard NER.

Variants of “shared components” generally achieve higher performance than the independent modeling methods (“N models”), which demonstrates the usefulness of enabling knowledge sharing among different subtasks.

We also notice that BERT and BERT-CRF models get lower performance than their BiLSTM and BiLSTM-CRF counterparts. The reason could be the domain discrepancy between the corpora that BERT is pretrained on and the product title/bullets. The former consist of mainly natural language sentences, while the latter are made up of integration of keywords and ungrammatical sentences.

5.2 High- vs. Low-Resource Attributes

To better understand the gain achieved by joint modeling, we further split the 12 testing attributes into 8 high-resource attributes and 4 low-resource attributes, based on the size of the training data with 1000 instances as the threshold. It is important to point out that many factors (e.g., vocabulary size, value ambiguity, and domain diversity), other than the size of training data, can contribute to the difficulty of modeling an attribute. Therefore, the performance for different attributes is not directly comparable.  

From results in Table 5, we can see that our model gets a lot more significant improvement from the independent modeling approach (BiLSTM-CRF (N models)) on low-resource attributes compared to high-resource attributes. This suggests that low-resource attributes benefit more from knowledge sharing, making our model desirable in the real-world setting with imbalanced attribute distribution.

5.3 Ablation Studies

Attribute Embeddings. We study different choices of adopting pretrained attribute embeddings as the initialization of the word embedding matrix \( W_\text{word} \). We choose \( d_h = 200 \) as the hidden size of the BiLSTM layer and 32 as the batch size. BERT-based models are optimized using AdamW (Loshchilov and Hutter, 2019) optimizer with learning rate \( 2e^{-5} \). Others use the Adam (Kingma and Ba, 2015) optimizer with learning rate \( 1e^{-3} \). We perform early stopping if no improvement in (Macro-) \( F_1 \) is observed on the development set for 3 epochs. For our model, we use contextualized attribute embeddings as described in §3.2 and freeze them during training. We set \( k = 3 \) for MoE. We made choices based on the development set performance.
dings. Specially, we experiment with contextualized embeddings (BERT name+value) and uncontextualized embeddings (Glove name+value) under the “Title” setting. For given attribute embeddings, we can either fine-tune them during training or freeze them once loaded. We also experiment with attribute name embeddings $r_{\text{name}}$ and attribute value embeddings $r_{\text{value}}$ only to understand which information is more helpful. The baseline is set as using randomly initialized learnable attribute embeddings. Table 6 shows the results. Comparing attribute embeddings with the same dimension, we find that freezing pretrained embeddings always leads to performance gain over the random baseline. This is because our parameterization methods have high flexibility in generating the parameters for the decoder. Using pretrained embeddings and freezing them provides the model with a good starting point and makes learning easier by reducing the degree of freedom. BERT name (freeze) outperforms BERT value (freeze), suggesting that the attribute name is more informative in determining the characteristics of the value extraction task on our dataset, where the values labeled through distant supervision are noisy.

<table>
<thead>
<tr>
<th>Attribute Embeddings</th>
<th>Dimension</th>
<th>P(%)</th>
<th>R(%)</th>
<th>F1(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>100</td>
<td>63.05</td>
<td>72.35</td>
<td>64.82</td>
</tr>
<tr>
<td>Glove name+value</td>
<td>100</td>
<td>64.12</td>
<td>70.51</td>
<td>63.89</td>
</tr>
<tr>
<td>Glove name+value (freeze)</td>
<td>100</td>
<td>64.47</td>
<td>73.11</td>
<td>65.53</td>
</tr>
<tr>
<td>Random</td>
<td>768</td>
<td>63.83</td>
<td>72.39</td>
<td>65.12</td>
</tr>
<tr>
<td>BERT name</td>
<td>768</td>
<td>62.01</td>
<td>73.94</td>
<td>64.89</td>
</tr>
<tr>
<td>BERT name (freeze)</td>
<td>768</td>
<td>64.90</td>
<td>74.31</td>
<td>66.60</td>
</tr>
<tr>
<td>BERT value</td>
<td>768</td>
<td>65.03</td>
<td>72.36</td>
<td>65.53</td>
</tr>
<tr>
<td>BERT value (freeze)</td>
<td>768</td>
<td>62.96</td>
<td>73.92</td>
<td>65.51</td>
</tr>
<tr>
<td>Random</td>
<td>1536</td>
<td>64.80</td>
<td>71.95</td>
<td>65.74</td>
</tr>
<tr>
<td>BERT name+value</td>
<td>1536</td>
<td>63.57</td>
<td>73.57</td>
<td>65.81</td>
</tr>
<tr>
<td>BERT name+value (freeze)</td>
<td>1536</td>
<td>65.00</td>
<td>75.87</td>
<td>67.48</td>
</tr>
</tbody>
</table>

Table 6: Performance (Title) with different choices for deriving and adopting attribute embeddings.

<table>
<thead>
<tr>
<th>Linear Layer</th>
<th>Transition Matrix</th>
<th>P(%)</th>
<th>R(%)</th>
<th>F1(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MoE</td>
<td>MoE</td>
<td>42.28</td>
<td>65.80</td>
<td>47.94</td>
</tr>
<tr>
<td>hypernetwork</td>
<td>hypernetwork</td>
<td>65.59</td>
<td>69.39</td>
<td>63.66</td>
</tr>
<tr>
<td>MoE</td>
<td>hypernetwork</td>
<td>53.52</td>
<td>66.43</td>
<td>55.10</td>
</tr>
<tr>
<td>hypernetwork</td>
<td>MoE</td>
<td>65.00</td>
<td>75.87</td>
<td>67.48</td>
</tr>
</tbody>
</table>

Table 7: Performance (Title) with different parameterization methods.

5.4 Effect of Number of Attributes

An important motivation of our model is that joint modeling of different attributes can facilitate knowledge sharing and improve the performance. Here we study the performance of model improvement along with increment of the number of jointly modeled attributes. We experiment under the “Title” setting. We start with training our model on 12 attributes that have test data. After that, we random select 5, 10, 15, 20 attributes from the remaining attributes, and add them to the joint training. The evaluation results on 12 test attributes are presented in Figure 3. While our model general demonstrates greater improvement with joint modeling of more attributes, other models’ performance fluctuate or goes down. That also demonstrates the scalability
of our model when new attributes keep emerging in real-world scenarios.

6 Related Work

Attribute Value Extraction. OpenTag (Zheng et al., 2018) formulates attribute value extraction as a sequence tagging task, and proposes a BiLSTM-SelfAttention-CRF architecture to address the problem. Xu et al. (2019) propose an “attribute-aware” setup, by utilizing one set of BIO tags and attribute name embedding with an attention mechanism, to enforce the extraction network to be attribute comprehensive. Karamanolakis et al. (2020) additionally incorporate the product taxonomy into a multi-task learning setup, to capture the nuances across different product types. Zhu et al. (2020) introduce a multi-modal network to combine text and visual information with a cross-modality attention to leverage image rich information that is not conveyed in text. Wang et al. (2020) use a question answering formulation to tackle attribute value extraction. We adopt the extraction setup in our model as most of previous contributions, using sequence labeling architecture. But we utilize an adaptive decoding approach, where the decoding network is parameterized with the attribute embedding.

Dynamic Parameter Generation. Our model proposes an adaptive-based decoding setup, parameterized with attribute embeddings through a Mixture-of-Experts module and a hypernetwork. Jacobs et al. (1991) first propose a system composed of several different “expert” networks and use a gating network that decides how to assign different training instances to different “experts”. Alshaikh et al. (2020); Guo et al. (2018); Le et al. (2016); Peng et al. (2019) all use domain/knowledge experts, and combine the predictions of each expert with a gating network. Unlike these works, we combine the weights of each expert to parameterize a network layer given an input embedding. Ha et al. (2017) propose the general idea of generating the parameters of a network by another network. The proposed model in Cai et al. (2019) generates the parameters of an encoder-decoder architecture by referring to the context-aware and topic-aware input. Suarez (2017) uses a hypernetwork to scale the weights of the main recurrent network. Platanios et al. (2018) tackle neural machine translation between multiple languages using a universal model with a contextual parameter generator.

7 Conclusion

In this work we propose a multi-attribute value extraction model that performs joint modeling of many attributes using an adaptive CRF-based decoder. Our model has a high capacity to derive attribute-specific network parameters while facilitating knowledge sharing. Incorporated with pre-trained attribute embeddings, our model shows marked improvements over previous methods.

Acknowledgments

This work has been supported in part by NSF SMA 18-29268. We would like to thank Jun Ma, Chenwei Zhang, Colin Lockard, Pascual Martínez-Gómez, Binxuan Huang from Amazon, and all the collaborators in USC INK research lab, for their constructive feedback on the work. We would also like to thank the anonymous reviewers for their valuable comments.

References


A Number of Model Parameters

<table>
<thead>
<tr>
<th>Methods</th>
<th># Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>BiLSTM (N tag sets)</td>
<td>0.6k · N + 6M</td>
</tr>
<tr>
<td>BiLSTM-CRF (N tag sets)</td>
<td>9 · N^2 + 0.6k · N + 6M</td>
</tr>
<tr>
<td>BiLSTM/BiLSTM-CRF (N models)</td>
<td>6M · N</td>
</tr>
<tr>
<td>BiLSTM-CRF-SharedEmb</td>
<td>0.1M · N + 6M</td>
</tr>
<tr>
<td>BiLSTM-MultiCRF</td>
<td>2k · N + 6M</td>
</tr>
<tr>
<td>AdaTag</td>
<td>8M</td>
</tr>
</tbody>
</table>

Table 8: Numbers of parameters for BiLSTM-based models with N attributes.

In our main experiment (Table 4), the numbers of parameters (M = 1,000,000; k = 1,000) for BiLSTM-based models with N attributes are listed in Table 8. BERT (bert-base-cased) itself has 110M parameters, making BERT-based models generally much larger.

For our AdaTag, the weights for the hypernetwork (W_{hyper} ∈ \mathbb{R}^{4d_h × d_r}) have (4 × 200) × 1536 parameters. The number can be reduced by inserting a middle layer with fewer neurons.