Learning What’s Easy: Fully Differentiable Neural Easy-First Taggers

André F. T. Martins
Unbabel & Instituto de Telecomunicações
Lisbon, Portugal
andre.martins@unbabel.com

Julia Kreutzer*
Computational Linguistics
Heidelberg University, Germany
kreutzer@cl.uni-heidelberg.de

Abstract

We introduce a novel neural easy-first decoder that learns to solve sequence tagging tasks in a flexible order. In contrast to previous easy-first decoders, our models are end-to-end differentiable. The decoder iteratively updates a “sketch” of the predictions over the sequence. At its core is an attention mechanism that controls which parts of the input are strategically the best to process next. We present a new constrained softmax transformation that ensures the same cumulative attention to every word, and show how to efficiently evaluate and backpropagate over it. Our models compare favourably to BILSTM taggers on three sequence tagging tasks.

1 Introduction

In the last years, neural models have led to major advances in several structured NLP problems, including sequence tagging (Plank et al., 2016; Lample et al., 2016), sequence-to-sequence prediction (Sutskever et al., 2014), and sequence-to-tree (Dyer et al., 2015). Part of the success comes from clever architectures such as (bidirectional) long-short term memories (BILSTMs; Hochreiter and Schmidhuber (1997); Graves et al. (2005)) and attention mechanisms (Bahdanau et al., 2015), which are able to select the pieces of context relevant for prediction.

A noticeable aspect about many of the systems above is that they typically decode from left to right, greedily or with a narrow beam. While this is computationally convenient and reminiscent of the way humans process spoken language, the combination of unidirectional decoding and greediness leads to error propagation and suboptimal classification performance. This can partly be mitigated by globally normalized models (Andor et al., 2016) and imitation learning (Daumé et al., 2009; Ross et al., 2011; Bengio et al., 2015), however these techniques still have a left-to-right bias.

Easy-first decoders (Tsuruoka and Tsujii, 2005; Goldberg and Elhadad, 2010, §2) are an interesting alternative: instead of a fixed decoding order, these methods schedule their own actions by preferring “easier” decisions over more difficult ones. A disadvantage is that these models are harder to learn, due to the factorial number of orderings leading to correct predictions. Usually, gradients are not backpropagated over this combinatorial latent space (Kiperwasser and Goldberg, 2016), or a separate model is used to determine the easiest next move (Clark and Manning, 2016).

In this paper, we develop novel, fully differentiable, neural easy-first sequence taggers (§3). Instead of taking discrete actions, our decoders use an attention mechanism to decide (in a soft manner) which word to focus on for the next tagging decision. Our models are able to learn their own sense of “easiness”: the words receiving focus may not be the ones the model is most confident about, but the best to avoid error propagation in the long run. To make sure that all words receive the same cumulative attention, we further contribute with a new constrained softmax transformation (§4). This transformation extends the softmax by permitting upper bound constraints on the amount of probability a word can receive. We show how to evaluate this transformation and backpropagate its gradients.

We run experiments in three sequence tagging tasks: multilingual part-of-speech (POS) tagging, named entity recognition (NER), and word-level quality estimation (§5). We complement our findings with a visual analysis of the attention distribu-
Algorithm 1 Easy-First Sequence Tagging

Input: input sequence $x_{1:L}$
Output: tagged sequence $y_{1:L}$
1: initialize $B = S = \emptyset$
2: while $B \neq \{1, \ldots, L\}$ do
3: for each non-covered position $i \notin B$ do
4: compute scores $f(i, y_i; x_{1:L}, S)$, $\forall y_i$
5: end for
6: $(j, \hat{y}_j) = \arg\max_{i, y_i} f(i, y_i; x_{1:L}, S)$
7: $S \leftarrow S \cup \{(j, \hat{y}_j)\}$, $B \leftarrow B \cup \{j\}$
8: end while

- The idea behind easy-first decoding is to perform “easier” and less risky decisions before committing to more difficult ones (Tsuruoka and Tsujii, 2005; Goldberg and Elhadad, 2010; Ma et al., 2013). Alg. 1 shows the overall procedure for a sequence tagging problem (the idea carries out to other structured problems). Let $x_{1:L}$ be an input sequence (e.g. words in a sentence) and $y_{1:L}$ be the corresponding tag sequence (e.g. their POS tags). The algorithm assigns tags one position at a time, maintaining a set $B$ of covered positions. It also maintains a set $S$ of pairs $(i, \hat{y}_i)$, storing the tags that have already been predicted at those positions. We can regard this set as a sketch of the output sequence, built incrementally while the algorithm is executed. At each time step, the model computes a score $f(i, y_i; x_{1:L}, S)$ for each position $i \notin B$ and each candidate tag $y_i$, taking into account the current “sketch” $S$, which provides useful contextual information. The “easiest” position and the corresponding tag are then jointly obtained by maximizing this score (line 6). The algorithm terminates when all positions are covered.

- Previous work has trained easy-first systems with variants of the perceptron algorithm (Goldberg and Elhadad, 2010; Ma et al., 2013) or with a gradient-based method (Kiperwasser and Goldberg, 2016)—but without backpropagating information about the best ordering chosen by the algorithm (only tag mistakes). In fact, doing so directly would be hard, since the space of possible orderings is combinatorial—the argmax in line 6 is not continuous, let alone differentiable. In the next section, we introduce a fully differentiable easy-first system that sidesteps this problem by working with a “continuous” space of actions.

2 Easy-First Decoders

- The high-level procedure is shown in Figure 1. The idea behind easy-first decoding is to perform “easier” and less risky decisions before committing to more difficult ones (Tsuruoka and Tsujii, 2005; Goldberg and Elhadad, 2010; Ma et al., 2013). Alg. 1 shows the overall procedure for a sequence tagging problem (the idea carries out to other structured problems). Let $x_{1:L}$ be an input sequence (e.g. words in a sentence) and $y_{1:L}$ be the corresponding tag sequence (e.g. their POS tags). The algorithm assigns tags one position at a time, maintaining a set $B$ of covered positions. It also maintains a set $S$ of pairs $(i, \hat{y}_i)$, storing the tags that have already been predicted at those positions. We can regard this set as a sketch of the output sequence, built incrementally while the algorithm is executed. At each time step, the model computes a score $f(i, y_i; x_{1:L}, S)$ for each position $i \notin B$ and each candidate tag $y_i$, taking into account the current “sketch” $S$, which provides useful contextual information. The “easiest” position and the corresponding tag are then jointly obtained by maximizing this score (line 6). The algorithm terminates when all positions are covered.

- Previous work has trained easy-first systems with variants of the perceptron algorithm (Goldberg and Elhadad, 2010; Ma et al., 2013) or with a gradient-based method (Kiperwasser and Goldberg, 2016)—but without backpropagating information about the best ordering chosen by the algorithm (only tag mistakes). In fact, doing so directly would be hard, since the space of possible orderings is combinatorial—the argmax in line 6 is not continuous, let alone differentiable. In the next section, we introduce a fully differentiable easy-first system that sidesteps this problem by working with a “continuous” space of actions.

Figure 1: A neural easy-first system applied to a POS tagging problem. Given the current input/sketch representation, an attention mechanism decides where to focus (see bar plot) and is used to generate the next sketch. Right: A sequence of sketches $(S^n)_{n=1}^N$ generated along the way.
Algorithm 2 Neural Easy-First Sequence Tagging

Input: input sequence $x_{1:L}$, sketch steps $N$
Output: tagged sequence $y_{1:L}$
1: initialize $\beta^0 = 0$ and $s^0_i = 0$, $\forall i \in [L]$
2: encode sequence as $[h_1, \ldots, h_L]$
3: for $n = 1, 2, \ldots, N$ do
4:  for each position $i \in [L]$ do
5:  compute $c^n_i$ and $s^n_i$ (Eqs. 1–2)
6:  end for
7:  compute attention $\alpha^n = \rho(z^n; \beta^{n-1})$
8:  for each position $i \in [L]$ do
9:  refine sketch $s^n_i$ from $\alpha^n_i$ and $c^n_i$, via Eq. 4 (single-state) or Eq. 7 (full-state)
10: end for
11: $\beta^n = \beta^{n-1} + \alpha^n$
12: end for
13: for each position $i \in [L]$ do
14:  $p_i = \text{softmax}(\text{affine}(\text{concat}(h_i, s^n_i)))$
15:  predict $\hat{y}_i = \text{argmax}_{p_i}(p_i)$
16: end for

$H = [h_1, \ldots, h_L] \in \mathbb{R}^{D_h \times L}$ (line 2). Our model is completely agnostic about this encoding step. In our experiments, we compute $H$ by composing a lookup embedding layer with a BILSTM (Hochreiter and Schmidhuber, 1997; Graves et al., 2005), as this strategy was successful in similar structured prediction tasks. However, other choices are possible, for example using convolutional layers.

As stated above, our algorithm maintains a cumulative attention vector $\beta \in \mathbb{R}^L$ and a sketch matrix $S \in \mathbb{R}^{D_s \times L}$, both with all entries initialized to zero. It then performs $N$ sketching steps, which progressively refine this sketch matrix, producing versions $S_1, \ldots, S_N$. At the $n$th step, the following operations are performed:

**Input-Sketch Contextual Representation.** For each word $i \in [L]$, we compute a state $c^n_i$ summarizing the surrounding local information about other words and sketches. We use a simple vector concatenation over a $w$-wide context window:

$$c^n_i = \text{concat}(h_{i-w}, \ldots, h_{i}, h_{i+w}, s^{n-1}_{i-w}, \ldots, s^{n-1}_{i+w}),$$

(1) where we denote by $s^{n-1}_j$ the $j$th column of $S^{n-1}$.

The intuition is that the current sketches provide valuable information about the neighboring words’ predictions that can influence the prediction for the $i$th word. In the vanilla easy-first algorithm, this was assumed in the score computation (line 4 of Alg. 1).

**Attention Mechanism.** We then use an attention mechanism to decide what is the “best” word to focus on next. This is done in a similar way as the feedforward attention proposed by Bahdanau et al. (2015). We first compute a score for each word $i \in [L]$ based on its contextual representation,

$$z^n_i = v^T \tanh(\text{affine}(c^n_i)),$$

(2) where $v \in \mathbb{R}^{D_s}$ is a model parameter. Then, we aggregate these scores in a vector $z^n \in \mathbb{R}^L$ and apply a transformation $\rho$ to map them to a probability distribution $\alpha^n \in \Delta^{L-1}$ (optionally taking into account the past cumulative attention $\beta^{n-1}$):

$$\alpha^n = \rho(z^n; \beta^{n-1}).$$

(3) The “standard” choice for $\rho$ is the softmax transformation. However, in this work, we consider other possible transformations (to be described in §4). After this, the cumulative attention is updated via $\beta^n = \beta^{n-1} + \alpha^n$.

**Sketch Generation.** Now that we have a distribution $\alpha^n \in \Delta^{L-1}$ over word positions, it remains to generate a sketch for those words. We first compute a single-state vector representation of the entire sentence $\tilde{c}^n = \sum_{i=1}^L \alpha^n_i c^n_i$ as the weighted average of the word states defined in Eq. 1. Then, we update each column of the sketch matrix as:

$$s^n_i = s^{n-1}_i + \alpha^n_i \cdot \tanh(\text{affine}(\tilde{c}^n_i)), \quad \forall i \in [L].$$

(4) The intuition for this update is the following: in the extreme case where the attention distribution is peaked on a single word (say, the $k$th word, $\alpha^n_k = e_k$), we obtain $\tilde{c}^n = c^n_k$ and the sketch update only affects that word, i.e.,

$$s^n_i = \begin{cases} 
  s^{n-1}_i + \tanh(\text{affine}(c^n_k)) & \text{if } i = k \\
  s^{n-1}_i & \text{if } i \neq k.
\end{cases}$$

(5) This is similar to the sketch update in the original easy-first algorithm (line 7 in Alg. 1).

The three operations above are repeated $N$ times (or “sketch steps”). The standard choice is $N = L$ (one step per word), which mimics the vanilla easy-first algorithm. However, it is possible to have fewer steps (or more, if we want the decoder to be able to “self-correct”). After completing the $N$ sketch steps, we obtain the final sketch matrix $S^N = [s^N_1, \ldots, s^N_L]$. Then, we compute a tag probability for every word as follows:

$$p_i = \text{softmax}(\text{affine}(\text{concat}(h_i, s^N_i))).$$

(6)
In §5, we compare this to a BILSTM tagger, which predicts according to $p_n = \text{softmax}(\text{affine}(h_n))$.

## 3.2 Full-State Model (NEF-F)

The full-state model differs from the single-state model in §3.1 by computing a full matrix for every sketch step, instead of a single vector. Namely, instead of Eq. 4, it does the following sketch update for every word $i \in [L]$:

$$s_i^n = s_i^{n-1} + c_i^n \cdot \tanh(\text{affine}(e_i^n)). \quad (7)$$

Note that the only difference is in replacing the single vector $\bar{c}^n$ by the word-specific vector $c_i^n$. As a result, this is no longer a rank-one update of the sketch matrix, but a full update. In the extreme case where the attention is peaked on a single word, the sketch update reduces to the same form as in the single-state model (Eq. 5). However, the full-state model is generally more flexible and allows processing words in parallel, since it allows different sketch updates for multiple words receiving attention, instead of trying to force those words to receive the same update. We will see in the experiments (§5) that this flexibility can be important in practice.

### 3.3 Computational Complexity

For both models, assuming that the $\rho(z; \beta)$ transformation in Eq. 3 takes $O(L)$ time to compute, the total runtime of Alg. 2 is $O((N + K)L)$ (where $K$ is the number of tags), which becomes $O(L^2)$ if $K \leq N = L$. This is so because the input-sketch representation, the attention mechanism, and the sketch generation step all have $O(L)$ complexity, and the final softmax layer requires $O(KL)$ operations. This is the same runtime of the vanilla easy-first algorithm, though the latter can be reduced to $O(KL \log L)$ with caching and a heap, if the scores in line 4 depend only on local sketches (Goldberg and Elhadad, 2010). By comparison, a standard BILSTM tagger has runtime $O(KL)$.

## 4 Constrained Softmax Attention

An important part of our models is their attention component (line 7 in Alg. 2). To keep the “easy-first” intuition, we would like the transformation $\rho$ in Eq. 3 to have a couple of properties:

1. **Sparsity:** being able to generate sparse distributions $\alpha^n$ (ideally, peaked on a single word).

2. **Evenness:** over iterations, spreading attention evenly over the words. Ideally, the cumulative attention should satisfy $\beta^n \in [0, 1]^L$ for every $n \in [N]$ and $\beta^N = 1$.

The standard choice for attention mechanisms is the softmax transformation, $\alpha^n = \text{softmax}(z^n)$. However, the softmax does not satisfy either of the properties above. For the first requirement, we could incorporate a “temperature” parameter in the softmax to push for more peaked distributions. However, this does not guarantee sparsity (only “hard attention” in the limit) and we found it numerically unstable when plugged in Alg. 2. For the second one, we could add a penalty before the softmax transformation, $\alpha^n = \text{softmax}(z^n - \lambda \beta^{n-1})$, where $\lambda \geq 0$ is a tunable hyperparameter. This strategy was found effective to prevent a word to receive too much attention, but it made the model less accurate.

An alternative is the sparsemax transformation (Martins and Astudillo, 2016):

$$\text{sparsemax}(z) = \arg\min_{\alpha \in \Delta^{L-1}} \|\alpha - z\|^2. \quad (8)$$

The sparsemax maintains most of the appealing properties of the softmax (efficiency to evaluate and backpropagate), and it is able to generate truly sparse distributions. However, it still does not satisfy the “evenness” property.

Instead, we propose a novel constrained softmax transformation that satisfies both requirements. It resembles the standard softmax, but it allows imposing hard constraints on the maximal probability assigned to each word. Let us start by writing the (standard) softmax in the following variational form (Amari and Nagaoka, 2001):

$$\text{softmax}(z) = \arg\min_{\alpha \in \Delta^{L-1}} \text{KL}(\alpha || \text{softmax}(z)) = \arg\min_{\alpha \in \Delta^{L-1}} -H(\alpha) - z^\top \alpha, \quad (9)$$

where $\text{KL}$ and $H$ denote the Kullback-Leibler divergence and the entropy, respectively. Based on this observation, we define the constrained softmax transformation as follows:

$$\text{csoftmax}(z; u) = \arg\min_{\alpha \in \Delta^{L-1}} -H(\alpha) - z^\top \alpha \quad \text{s.t.} \quad \alpha \leq u, \quad (10)$$

where $u \in \mathbb{R}^L$ is a vector of upper bounds. Note that, if $u \geq 1$, all constraints are loose and this...
reduces to the standard softmax; on the contrary, if $u \in \Delta^{L-1}$, they are tight and we must have $\alpha = u$ due to the normalization constraint. Thus, we propose the following for Eq. 3:

$$\alpha^n = c\text{softmax}(z^n; 1 - \beta^{n-1}). \quad (11)$$

The constraints guarantee $\beta^n = \beta^{n-1} + \alpha^n \leq 1$. Since $\mathbf{1}^\top \mathbf{B}^N = \sum_{n=1}^{N} \mathbf{1}^\top \mathbf{\alpha}^n = N$, they also ensure that $\beta^N = 1$, hence the “evenness” property is fully satisfied. Intuitively, each word gets a credit of one unit of attention that is consumed during the execution of the algorithm. When this credit expires, all subsequent attention weights for that word will be zero.

The next proposition shows how to evaluate the constrained softmax and compute its gradients.

**Proposition 1** Let $\alpha = c\text{softmax}(z; u)$, and define the set $A = \{i \in [L] \mid \alpha_i < u_i\}$ of the constraints in Eq. 10 that are met strictly. Then:

- **Forward propagation.** The solution of Eq. 10 can be written in closed form as $\alpha_i = \min\{\exp(z_i)/Z, u_i\}$, where $Z = \sum_{i \in A} \exp(z_i) / (1 - \sum_{i \notin A} u_i)$.

- **Gradient backpropagation.** Let $L(\theta)$ be a loss function, $d\alpha = \nabla_{\alpha} L(\theta)$ be the output gradient, and $dz = \nabla_{z} L(\theta)$ and $du = \nabla_{u} L(\theta)$ be the input gradients. Then, we have:

$$dz_i = \mathbf{1}(i \in A) \alpha_i (d\alpha_i - m) \quad (12)$$

$$du_i = \mathbf{1}(i \notin A) (d\alpha_i - m) \quad (13)$$

where $m = (\sum_{i \in A} \alpha_i d\alpha_i) / (1 - \sum_{i \notin A} u_i)$.

**Proof:** See App. A (supplementary material). ■

Algs. 3–4 turn the results in Prop. 1 into concrete procedures for evaluating $c\text{softmax}$ and for backpropagating its gradient. Their runtimes are respectively $O(L \log L)$ and $O(L)$.

### 5 Experiments

We evaluate our neural easy-first models in three sequence tagging tasks: POS tagging, NER, and word quality estimation.

#### 5.1 Part-of-Speech Tagging

We ran POS tagging experiments in 12 languages from the Universal Dependencies project v1.4 (Nivre et al., 2016), using the standard splits. The datasets contain 17 universal tags.\(^3\)

We implemented Alg. 2 in DyNet (Neubig et al., 2017), which we extended with the constrained softmax operator (Algs. 3–4).\(^4\) We used 64-dimensional word embeddings, initialized with pre-trained Polyglot vectors (Al-Rfou et al., 2013). Apart from the words, we embedded prefix and suffix character $n$-grams with $n \leq 4$. We set the affix embedding size to 50 and summed all these embeddings; in the end, we obtained a 164-dimensional representation for each word (words, prefixes, and suffixes). We then fed these embeddings into a BiLSTM (with 50 hidden units in each direction) to obtain the encoder states $[h_1, \ldots, h_L] \in \mathbb{R}^{100 \times L}$. The other hyperparameters were set as follows: we used a context size $w = 2$, set the pre-attention size $D_c$ and the sketch size $D_s$ to 50, and applied dropout with a probability of 0.2 after the embedding and BiLSTM layers and before the final softmax output layer.\(^5\) We ran 20 epochs of Adagrad to minimize the cross-entropy loss, with a stepsize of 0.1, and gradient

\(^3\)Our choice of languages covers 8 families: Romance (French, Portuguese, Spanish), Germanic (English, German), Slavic (Czech, Russian), Semitic (Arabic), Indo-Iranian (Hindi), Austronesian (Indonesian), Sino-Tibetan (Chinese), and Japonic (Japanese). For all languages, we used the datasets that have the canonical language name, except for Russian, where we used the (much larger) SynTagRus corpus. We preferred large datasets over small ones, to reduce the impact of overfitting in our analysis.

\(^4\)The code is available at https://github.com/Unbabel/neural-easy-first.

\(^5\)These hyperparameters were tuned in the English development set, and kept fixed across languages.
Table 1: POS tagging accuracies. The average in the rightmost column is over the words of each treebank.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Gillick et al. (2016)†</td>
<td>–</td>
<td>–</td>
<td>98.44</td>
<td>94.00</td>
<td>95.17</td>
<td>92.34</td>
<td>–</td>
<td>–</td>
<td>91.03</td>
<td>–</td>
<td>–</td>
<td>95.26</td>
<td>–</td>
</tr>
<tr>
<td>Plank et al. (2016)†</td>
<td>98.91</td>
<td>98.24</td>
<td>95.16</td>
<td>96.11</td>
<td>93.38</td>
<td>97.10</td>
<td>93.41</td>
<td>–</td>
<td>97.90</td>
<td>–</td>
<td>–</td>
<td>95.74</td>
<td>–</td>
</tr>
<tr>
<td>Linear (TurboTagger)</td>
<td>95.18</td>
<td>91.38</td>
<td>98.07</td>
<td>94.43</td>
<td>96.48</td>
<td>91.92</td>
<td>95.98</td>
<td>93.12</td>
<td>89.35</td>
<td>96.63</td>
<td>97.32</td>
<td>95.44</td>
<td>94.88</td>
</tr>
<tr>
<td>BILSTM</td>
<td>95.60</td>
<td>92.73</td>
<td>98.46</td>
<td>94.94</td>
<td>96.37</td>
<td>92.97</td>
<td>96.80</td>
<td>93.79</td>
<td>92.72</td>
<td>97.01</td>
<td>97.83</td>
<td>95.27</td>
<td>95.39</td>
</tr>
<tr>
<td>Vanilla Easy-First</td>
<td>95.62</td>
<td>92.78</td>
<td>98.50</td>
<td>94.78</td>
<td>96.35</td>
<td>92.91</td>
<td>96.81</td>
<td>93.68</td>
<td>92.68</td>
<td>97.90</td>
<td>95.74</td>
<td>95.42</td>
<td>95.42</td>
</tr>
<tr>
<td>NEF-S, cssoftmax</td>
<td>95.63</td>
<td>92.87</td>
<td>98.50</td>
<td>94.91</td>
<td>96.41</td>
<td>92.86</td>
<td>96.79</td>
<td>93.65</td>
<td>92.93</td>
<td>96.57</td>
<td>97.91</td>
<td>95.42</td>
<td>95.42</td>
</tr>
<tr>
<td>NEF-F, softmax</td>
<td>95.61</td>
<td>92.92</td>
<td>98.47</td>
<td>95.15</td>
<td>96.52</td>
<td>92.96</td>
<td>96.75</td>
<td>93.67</td>
<td>92.57</td>
<td>97.12</td>
<td>97.92</td>
<td>95.41</td>
<td>95.43</td>
</tr>
<tr>
<td>NEF-F, sparsemax</td>
<td>95.14</td>
<td>92.86</td>
<td>97.75</td>
<td>93.97</td>
<td>91.79</td>
<td>90.27</td>
<td>89.40</td>
<td>93.48</td>
<td>92.61</td>
<td>95.25</td>
<td>96.32</td>
<td>95.36</td>
<td>93.88</td>
</tr>
<tr>
<td>NEF-F, cssoftmax</td>
<td>95.53</td>
<td>92.99</td>
<td>98.53</td>
<td>95.01</td>
<td>96.70</td>
<td>92.93</td>
<td>96.98</td>
<td>93.81</td>
<td>92.72</td>
<td>97.04</td>
<td>97.94</td>
<td>95.42</td>
<td>95.47</td>
</tr>
</tbody>
</table>

†Note that Gillick et al. (2016) and Plank et al. (2016) are not strictly comparable, since they use older versions of the treebanks (UD1.1 and UD1.2, respectively).

Figure 2: Attention visualization for English POS tagging. From the left: constrained softmax, softmax, sparsemax, vanilla. Rows correspond to attention vectors (high values in yellow, low ones in dark blue).

clipping of 5 (DyNet’s default). We excluded from the training set sentences longer than 50 words.

Table 1 compares several variants of our neural easy-first system—the single-state model (NEF-S), the full-state model (NEF-F), and the latter with softmax, sparsemax, and cssoftmax attention. We used as many sketch steps as the number of words, \( N = L \). As baselines, we used:

- A feature-based linear model (TurboTagger, Martins et al. (2013)).
- A BILSTM tagger identical to our system, but without sketch steps (\( N = 0 \)).
- A vanilla easy-first tagger (Alg. 1), using the argument of the softmax in Eq. 6 as the scoring function. This uses the same sketch representations as the neural easy-first systems, but replaces the attention mechanism by “hard” attention placed on the highest scored word.

For comparison, we also show the accuracies reported by Gillick et al. (2016) for their byte-to-span system (trained separately on each language) and by Plank et al. (2016) for their state-of-the-art multi-task BILSTM tagger (these results are not fully comparable though, due to different treebank versions).

Among the neural easy-first systems, we observe that NEF-F with cssoftmax attention generally outperforms the others, but the differences are very slight (excluding the sparsemax attention system, which performed substantially worse). This system wins over the linear system for all languages but Spanish, and over the BILSTM baseline for 9 out of 12 languages (loses in Arabic and German, and ties in Japanese). Note, however, that the differences are small (95.47% against 95.39%, averaged across treebanks). We conjecture that this is due to the fact that the BILSTM already captures most of the relevant context in its encoder. Our NEF-F system with cssoftmax also wins over the vanilla easy-first system for 10 out of 12 languages (arguably due to its ability to backpropagate the gradients through the soft attention mechanism), but the difference in the average score is again small (95.47% against 95.42%).

Figure 2 depicts some patterns learned by the NEF-F model with various attention types. With the cssoftmax, the model learns to move left and right, and the main verb “thought” is the most
prominent candidate for the easiest decision. In fact, in 57% of the test sentences, the model focuses first on a verb. The “raindrop” appearance of the plot is due to the evenness property of csoftmax, which causes the attention over a word to increase gradually until the cumulative attention is exhausted. This contrasts with the softmax attention (less diverse and non-sparse) and the sparsemax (sparse, but not even). We show for comparison the (hard) decisions made by the vanilla easy-first decoder.

5.2 Named Entity Recognition

Next, we applied our model to NER. We used the official datasets from the CoNLL 2002-3 shared tasks (Tjong Kim Sang, 2002; Tjong Kim Sang and De Meulder, 2003), which tag names, locations, and organizations using a BIO scheme, and cover four languages (Dutch, English, German, and Spanish). We made two experiments: one using the exact same BILSTM and NEF models with a standard softmax output layer, as in §5.1 (which does not guarantee valid segmentations), and another one replacing the output softmax layer by a sequential CRF layer, which requires learning $O(K^2)$ additional parameters for pairs of consecutive tags (Huang et al., 2015; Lample et al., 2016). We used the same hyperparameters as in the POS tagging experiments, except the dropout probability, which was set to 0.3 (tuned on the validation set). For English, we used pre-trained 300-dimensional GloVe-840B embeddings (Pennington et al., 2014); for Spanish and German, we used the 64-dimensional word embeddings from Lample et al. (2016); for Dutch we used the aforementioned Polyglot vectors. All embeddings are fine-tuned during training. Since many words are not entities, and those receive a default “outside” tag, we expect that fewer sketch steps are necessary to achieve top performance.

Table 2 shows the results, which confirm this hypothesis. We compare the same BILSTM baseline to our NEF-S and NEF-F models with csoftmax attention (with and without the CRF output layer), varying the maximum number of sketch steps. We also compare against the byte-to-span model of Gillick et al. (2016) and the state-of-the-art character-based LSTM-CRF system of Lample et al. (2016).\(^6\) We can see that, for all languages, the NEF-CRF-F model with 5 steps is consistently better than the BILSTM-CRF and, with the exception of English, the NEF-CRF-S model. The same holds for the BILSTM and NEF-F models without the CRF output layer. With the exception of German, increasing the number of steps did not make a big difference.

Figure 3 shows the attention distributions over the sketch steps for an English sentence, for full-state models trained with $N \in \{5, L\}$. The model with $L$ sketch steps learned that it is easiest to focus on the beginning of a named entity, and then to move to the right to identify the full span. The model with only 5 sketch steps learns to go straight to the point, placing most attention on the entity words and ignoring most of the o-tokens.

5.3 Word-Level Quality Estimation

Finally, we evaluate our model’s performance on word-level translation quality estimation. The goal is to evaluate a translation system’s quality without access to reference translations (Blatz et al., 2004; Specia et al., 2013). Given a sentence pair (a source sentence and its machine translated sentence in a target language), a word-level system classifies each target word as OK or BAD. We used the official English-German dataset from the WMT16 shared task (Bojar et al., 2016).

This task differs from the previous ones in which its input is a sentence pair and not a single sentence, mixing character and word-based models, sharing a model across languages, or combining CRFs with convolutional and recurrent layers. We used simpler models in our experiments since our goal is to assess how much the neural easy-first systems can bring in addition to a BILSTM system, rather than building a state-of-the-art system.

\(^6\)The current state of the art on these datasets (Gillick et al., 2016; Lample et al., 2016; Ma and Hovy, 2016) is achieved by more sophisticated systems with more parameters, mixing character and word-based models, sharing a model across languages, or combining CRFs with convolutional and recurrent layers. We used simpler models in our experiments since our goal is to assess how much the neural easy-first systems can bring in addition to a BILSTM system, rather than building a state-of-the-art system.
Figure 3: Attention visualization for English NER, for 5 (top) and L (bottom) sketch steps. Words tagged as B-* are marked in light blue, those with I-* tags, in bold red, and with O-* tags, in green.

Figure 4: Example for word-level quality estimation. The source sentence is “To open the Actions panel, from the main menu, choose Window > Actions.” BAD words are red (bold font), OK words are green.

Table 3: $F_1$-MULT scores (product of $F_1$ for OK and BAD words) for word-level quality estimation, computed by the official shared task script.

<table>
<thead>
<tr>
<th></th>
<th>BILSTM</th>
<th>NEF-S</th>
<th>NEF-F</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N = 5$</td>
<td>39.71</td>
<td>40.91</td>
<td>41.18</td>
</tr>
<tr>
<td>$N = L$</td>
<td>40.99</td>
<td>40.84</td>
<td></td>
</tr>
</tbody>
</table>

Table 4 shows the results. As expected, the neural easy-first system was the best performing one, although the difference with respect to the ablated

5.4 Ablation Study

To better understand our proposed model, we carried out an ablation study for NER on the English dataset. The following alternate configurations were tried and compared against the NEF-CRF-F model with $csoftmax$ attention and 5 sketch steps:

- A NEF-CRF-F model for which the final concatenation in Eq. 6 was removed, being replaced by $p_i = \text{softmax}(\text{affine}(s_N^i))$. The goal was to see if the sketches retain enough information about the input to make a final prediction without requiring the states $h_i$.
- A model for which the attention mechanism applied at each sketch step was replaced by a uniform distribution over the input words.
- A vanilla easy-first system (Alg. 1). Since this system can only focus on one word at the time (unlike the models with soft attention), we tried both $N = 5$ and $N = L$ sketch steps.
- A left-to-right and right-to-left model, which replaces the attention mechanism by one of these two prescribed orders.

Our best system would rank third in the shared task, out of 13 submissions. The winner system, which achieved 49.52 $F_1$-MULT, was considerably more complex than ours, using an ensemble of three neural networks with a linear system (Martins et al., 2016).
Table 4: Ablation experiments. Reported are $F_1$ scores for NER in the English test set.

<table>
<thead>
<tr>
<th>System</th>
<th>$F_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEF-CRF-F, $N = 5$</td>
<td>88.01</td>
</tr>
<tr>
<td>NEF-CRF-F w/out concat, $N = 5$</td>
<td>87.47</td>
</tr>
<tr>
<td>Uniform Attention, $N = 5$</td>
<td>87.46</td>
</tr>
<tr>
<td>Vanilla EF + CRF, $N = 5$</td>
<td>87.17</td>
</tr>
<tr>
<td>Vanilla EF + CRF, $N = L$</td>
<td>87.46</td>
</tr>
<tr>
<td>Left-to-right + CRF, $N = L$</td>
<td>87.57</td>
</tr>
<tr>
<td>Right-to-left + CRF, $N = L$</td>
<td>87.53</td>
</tr>
</tbody>
</table>

Many systems is relatively small. Removing the concatenation in Eq. 6 is harmful, which suggests that there is information about the input not retained in the sketches. The uniform attention performs surprisingly well, and so do the left-to-right and right-to-left models, but they are still about half a point behind. The vanilla easy-first system has the worst performance with $N = 5$. This is due to the fact that the vanilla model is uncapable of processing words “in parallel” in the same sketch step, a disadvantage with respect to the neural easy-first models, which have this capability due to their soft attention mechanisms (see the top image in Fig. 3).

6 Related Work

Vanilla easy-first decoders have been used in POS tagging (Tsuruoka and Tsujii, 2005; Ma et al., 2013), dependency parsing (Goldberg and Elhadad, 2010), and coreference resolution (Stoyanov and Eisner, 2012), being related to cyclic dependency networks and guided learning (Toutanova et al., 2003; Shen et al., 2007). More recent works compute scores with a neural network (Socher et al., 2011; Clark and Manning, 2016; Kiperwasser and Goldberg, 2016), but they still operate in a discrete space to pick the easiest actions (the non-differentiable argmax in line 6 of Alg. 1). Generalizing this idea to “continuous” operations is at the very core of our paper, allowing gradients to be fully backpropagated. In a different context, building differentiable computation structures has also been addressed by Graves et al. (2014); Grefenstette et al. (2015).

An important contribution of our paper is the constrained softmax transformation. Others have proposed alternatives to softmax attention, including the sparsemax (Martins and Astudillo, 2016) and multi-focal attention (Globerson et al., 2016). The latter computes a KL projection onto a budget polytope to focus on multiple words. Our constrained softmax also corresponds to a KL projection, but (i) it involves box constraints instead of a budget, (ii) it is normalized to 1, and (iii) we also backpropagate the gradient over the constraint variables. It also achieves sparsity (see the “raindrop” plots in Figures 2–4), and is suitable for sequentially computing attention distributions when diversity is desired (e.g. soft 1-to-1 alignments). Recently, Chorowski and Jaitly (2016) developed an heuristic with a threshold on the total attention as a “coverage criterion” (see their Eq. 11), however their heuristic is non-differentiable.

Our sketch generation step is similar in spirit to the “deep recurrent attentive writer” (DRAW, Gregor et al., 2015) which generates images by iteratively refining sketches with a recurrent neural network (RNN). However, our goal is very different: instead of generating images, we generate vectors that lead to a final sequence tagging prediction.

Finally, the visualization provided in Figures 2–4 brings up the question how to understand and rationalize predictions by neural network systems, addressed by Lei et al. (2016). Their model, however, uses a form of stochastic attention and it does not perform any iterative refinement like ours.

7 Conclusions

We introduced novel fully-differentiable easy-first taggers that learn to make predictions over sequences in an order that is adapted to the task at hand. The decoder iteratively updates a sketch of the predictions by interacting with an attention mechanism. To spread attention evenly through all words, we introduced a new constrained softmax transformation, along with an algorithm to backpropagate its gradients. Our neural-easy first decoder consistently outperformed a BILSTM on a range of sequence tagging tasks.

A natural direction for future work is to go beyond sequence tagging (which we regard as a simple first step) toward other NLP structured prediction problems, such as sequence-to-sequence prediction. This requires replacing the sketch matrix in Alg. 2 by a dynamic memory structure.

Acknowledgments

We thank the Unbabel research team and the reviewers for their comments. This work was supported by the Fundação para a Ciência e Tecnologia through contracts UID/EEA/50008/2013, PTDC/EEI-SII/7092/2014 (LearnBig), and CMUPERI/TIC/0046/2014 (GoLocal).
References


Neural architectures for named entity recognition. In Proc. of the Annual Meeting of the North-
American Chapter of the Association for Computational Linguistics.

Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2016. 

Joakim Nivre, Željko Agić, Lars Ahrenberg, Maria Je-
Graham Neubig, Chris Dyer, Yoav Goldberg, Austin 
xuezhe Ma, and Eduard Hovy. 2016. End-to-end se-
André F. T. Martins and Ramón Astudillo. 2016. From 
ning and dependency parsing with beam search. In In Proceedings of the 51st Annual Meeting of the 
Association for Computational Linguistics (Volume 2: Short Papers).

Xuezhe Ma and Eduard Hovy. 2016. End-to-end se-

André F. T. Martins, Miguel B. Almeida, and Noah A. 
non-projective turbo parsers. In Proc. of the Annual 
Meeting of the Association for Computational Lin-
guistics.

André F. T. Martins and Ramón Astudillo. 2016. From 
Softmax to Sparsemax: A Sparse Model of Atten-

Barbara Plank, Anders Sogaard, and Yoav Goldberg. 
2016. Multilingual part-of-speech tagging with 


Barbara Plank, Anders Sogaard, and Yoav Goldberg. 
2016. Multilingual part-of-speech tagging with 

Stéphane Ross, Geoffrey J Gordon, and Drew Bagnell. 
2011. A reduction of imitation learning and struc-
tured prediction to no-regret online learning. In AIS-
TATS, volume 1, page 6.

Linbin Shen, Giorgio Satta, and Aravind Joshi. 2007. 
Guided learning for bidirectional sequence classifi-

Richard Socher, Cliff C Lin, Chris Manning, and 
Andrew Y Ng. 2011. Parsing natural scenes and natu-
ral language with recursive neural networks. In Pro-
cedings of the 28th international conference on ma-


A Proof of Proposition 1

We provide here a detailed proof of Proposition 1.

A.1 Forward Propagation

The optimization problem is

\[ \text{csoftmax}(z, u) = \arg\min \quad -H(\alpha) - z^\top \alpha \]
\[ \text{s.t.} \begin{cases} 1^\top \alpha = 1 \\ 0 \leq \alpha \leq u. \end{cases} \]

The Lagrangian function is:

\[ L(\alpha, \lambda, \mu, \nu) = -H(\alpha) - z^\top \alpha + \lambda(1^\top \alpha - 1) - \mu^\top \alpha + \nu^\top (\alpha - u). \] (14)

To obtain the solution, we invoke the Karush-Kuhn-Tucker conditions. From the stationarity condition, we have

\[ 0 = \log(\alpha) + 1 - z + \lambda - \mu + \nu, \]

which due to the primal feasibility condition implies that the solution is of the form:

\[ \alpha = \exp(z + \mu - \nu)/Z, \] (15)

where \( Z \) is a normalization constant. From the complementarity slackness condition, we have that \( 0 < \alpha_i < u_i \) implies \( \mu_i = \nu_i = 0 \) and therefore \( \alpha_i = \exp(z_i)/Z \). On the other hand, \( \nu_i > 0 \) implies \( \alpha_i = u_i \). Hence the solution can be written as \( \alpha_i = \min\{\exp(z_i)/Z, u_i\} \), where \( Z \) is determined such that the distribution normalizes:

\[ Z = \frac{\sum_{i \in A} \exp(z_i)}{1 - \sum_{i \notin A} u_i}, \] (16)

with \( A = \{i \in [L] \mid \alpha_i < u_i\} \).

A.2 Gradient Backpropagation

We now turn to the problem of backpropagating the gradients through the constrained softmax transformation. For that, we need to compute its Jacobian matrix, i.e., the derivatives \( \frac{\partial \alpha_i}{\partial z_j} \) and \( \frac{\partial \alpha_i}{\partial u_j} \) for \( i, j \in [L] \).

Let us first express \( \alpha_i \) as

\[ \alpha_i = \begin{cases} \exp(z_i)(1-s) \\ \sum_{j \in A} \exp(z_j) \\ \alpha_i \\ u_i, \end{cases} \] (17)

where \( s = \sum_{j \notin A} u_j \). Note that we have \( \partial s/\partial z_j = 0, \forall j \), and \( \partial s/\partial u_j = 1(j \notin A) \). To compute the entries of the Jacobian matrix, we need to consider several cases.

Case 1: \( i \in A \). In this case, the evaluation of Eq. 17 goes through the first branch. Let us first compute the derivative with respect to \( u_j \). Two things can happen: if \( j \in A \), then \( s \) does not depend on \( u_j \), hence \( \frac{\partial \alpha_i}{\partial u_j} = 0 \). Else, if \( j \notin A \), we have

\[ \frac{\partial \alpha_i}{\partial u_j} = \frac{-\exp(z_i) \frac{\partial s}{\partial u_j}}{\sum_{k \in A} \exp(z_k)} = -\alpha_i/(1 - s). \]

Now let us compute the derivative with respect to \( z_j \). Three things can happen: if \( j \in A \) and \( i \neq j \), we have

\[ \frac{\partial \alpha_i}{\partial z_j} = \frac{-\exp(z_i) \exp(z_j)(1-s)}{\left(\sum_{k \in A} \exp(z_k)\right)^2} = -\alpha_i \alpha_j/(1 - s). \] (18)
If $j \in \mathcal{A}$ and $i = j$, we have
\[
\frac{\partial \alpha_i}{\partial z_j} = (1 - s) \times \frac{\exp(z_i) \sum_{k \in \mathcal{A}} \exp(z_k) - \exp(z_i)^2}{(\sum_{k \in \mathcal{A}} \exp(z_k))^2} = \alpha_i - \frac{\alpha_i^2}{(1 - s)}.
\] 
(19)

Finally, if $j \notin \mathcal{A}$, we have $\frac{\partial \alpha_i}{\partial z_j} = 0$.

**Case 2: $i \notin \mathcal{A}$** In this case, the evaluation of Eq. 17 goes through the second branch, which means that $\frac{\partial \alpha_i}{\partial z_j} = 0$, always. Let us now compute the derivative with respect to $u_j$. This derivative is always zero unless $i = j$, in which case $\frac{\partial \alpha_i}{\partial u_j} = 1$.

To sum up, we have:
\[
\frac{\partial \alpha_i}{\partial z_j} = \begin{cases} 
1 (i = j) \alpha_i - \frac{\alpha_i \alpha_j}{1 - s}, & \text{if } i, j \in \mathcal{A} \\
0, & \text{otherwise,}
\end{cases}
\]
and
\[
\frac{\partial \alpha_i}{\partial u_j} = \begin{cases} 
-\frac{\alpha_j}{1 - s}, & \text{if } i \in \mathcal{A}, j \notin \mathcal{A} \\
1, & \text{if } i, j \notin \mathcal{A}, i = j \\
0, & \text{otherwise.}
\end{cases}
\]

Therefore, we obtain:
\[
dz_j = \sum_i \frac{\partial \alpha_i}{\partial z_j} d\alpha_i
\]
\[
= \mathbf{1}(j \in \mathcal{A}) \left( \alpha_j d\alpha_j - \frac{\alpha_j \sum_{i \in \mathcal{A}} \alpha_i d\alpha_i}{1 - s} \right)
\]
\[
= \mathbf{1}(j \in \mathcal{A}) \alpha_j (d\alpha_j - m),
\]
(22)

and
\[
du_j = \sum_i \frac{\partial \alpha_i}{\partial u_j} d\alpha_i
\]
\[
= \mathbf{1}(j \notin \mathcal{A}) \left( d\alpha_j - \frac{\sum_{i \in \mathcal{A}} \alpha_i d\alpha_i}{1 - s} \right)
\]
\[
= \mathbf{1}(j \notin \mathcal{A}) (d\alpha_j - m),
\]
(23)

where $m = \frac{\sum_{i \in \mathcal{A}} \alpha_i d\alpha_i}{1 - s}$. 