Sequence Labeling and Transduction with Biased Actor-Critic Training of RNNs

Saeed Najafi[†] and **Colin Cherry**[‡] and **Grzegorz Kondrak**[†]

[†]Department of Computing Science
University of Alberta, Canada
{snajafi,gkondrak}@ualberta.ca

[‡]Google
Montreal, Canada
colin.a.cherry@gmail.com

Abstract

Neural approaches to sequence labeling often use a Conditional Random Field (CRF) to model their output dependencies, while Recurrent Neural Networks (RNN) are used for the same purpose in other tasks. We set out to establish RNNs as an attractive alternative to CRFs for sequence labeling. To do so, we address one of the RNN's most prominent shortcomings, the fact that it is not exposed to its own errors, by training with a biased Actor-Critic algorithm (AC-RNN). We comprehensively compare our AC-RNN with maximum likelihood training for both RNNs and CRFs on three structured-output tasks. The proposed AC-RNN efficiently matches the performance of the CRF on NER and CCG tagging, and outperforms it on machine transliteration. We also show that our biased Actor-Critic training is significantly better than other techniques for addressing exposure bias, such as scheduled sampling, and self-critical policy training.

1 Introduction

Sequence labeling is a canonical structured output problem, with many techniques available to track its chain-structured output dependencies. Conditional Random Fields (CRF) built over a neural feature layer have recently emerged as a best practice for sequence labeling tasks in NLP (Huang et al., 2015). Alternatively, one can track the same output dependencies using a Recurrent Neural Network (RNN) over the output sequence, similar to the decoder component in neural machine translation (NMT).

Using a decoder RNN instead of a CRF has several potential advantages, such as simplified implementation, tracking longer dependencies, and allowing for larger output vocabularies. However, shifting from the CRF's sequence-level training objective to the decoder RNN's sequence of token-level objectives may lead to suboptimal performance due to exposure bias. Exposure bias stems from the token-level maximum likelihood objective typically used in RNN training, which does not expose the model to its own errors (Bengio et al., 2015). RNNs are typically conditioned on gold-standard contexts during training, while at test time, the model is conditioned on its own predictions, creating a train-test mismatch. As sequence-level models, CRFs are immune to exposure bias.

We set out to establish the decoder RNN as an attractive alternative to the CRF for sequence labeling. We do so by adopting a simple and effective RNN training strategy to counter the exposure-bias problem, and by providing an experimental comparison to demonstrate that our modified RNN can match the CRF in accuracy, and surpass it in flexibility.

As our primary contribution, we propose a biased Actor-Critic reinforcement learning objective that specializes actor-critic training (Konda and Tsitsiklis, 2003) to sequence-labeling. Our complete system, dubbed as the AC-RNN, maintains the same neural feature layer that has proven so successful for the CRF, changing only the output layer.[1] We conduct a comprehensive analysis comparing the AC-RNN to the CRF under controlled conditions using a shared implementation. We demonstrate that on Named Entity Recognition (NER) and Combinatory Categorical Grammar (CCG) supertagging, the AC-RNN can match the performance of the CRF, while training more efficiently.

To demonstrate its flexibility, we also test our biased actor-critic training algorithm on machine transliteration, a monotonic transduction problem that straddles the boundary between sequence la-

[1]The tool will be made publicly available.

Figure 1: CCG supertagging of the sentence *'AirCanada Serves Toronto'*.

beling and full sequence-to-sequence modeling.

Finally, we compare biased Actor-Critic training with previous methods proposed to address exposure bias. On NER tagging, we empirically demonstrate that AC-RNN is significantly better than the RNN trained with scheduled sampling (Bengio et al., 2015). We also demonstrate that our approach is more suitable for sequence labeling and transduction tasks than other similar policy-gradient methods such as self-critical training of Rennie et al. (2017).

The paper is organized as follows. Related works are summarized in Section 2. Section 3 explains the base architecture of our models, providing the background information needed to introduce biased Actor-Critic training in Section 4. Our comprehensive experiments are presented in Section 5.

2 Prior Work

In sequence labeling, several neural methods have recently been shown to outperform earlier systems that use hand-engineered features. For the tasks of POS tagging, chunking and NER, Huang et al. (2015) apply a CRF output layer on top of a bidirectional RNN over the source. For the NER task, Lample et al. (2016) extend the RNN layer with character-level RNNs that capture information about word prefixes and suffixes. Ma and Hovy (2016) use Convolutional Neural Networks to a similar end. We build upon Lample et al.'s approach, replacing their CRF with a decoder RNN trained with a biased Actor-Critic objective.

We also apply the AC-RNN to CCG supertagging (Clark and Curran, 2004), where the model labels each word in a sentence with one of 1,284 morphosyntactic categories from CCGbank (Hockenmaier and Steedman, 2007); an example is shown in Figure 1. Bidirectional RNNs have been used in a number of recent supertagging systems (Lewis et al., 2016; Xu, 2016; Vaswani et al., 2016; Kadari et al., 2017; Wu et al., 2017). Among these, our effort is most similar to Vaswani et al. (2016), who also use an RNN decoder over a bidirectional encoder; however, they address exposure bias with scheduled sampling.

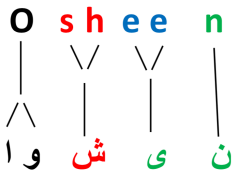

Figure 2: Transliteration of the name *'Osheen'* into Persian

We also consider transduction tasks, which go beyond sequence labeling by allowing many-to-many monotonic alignments between the source and target symbols. In particular, we focus on transliteration, where the goal is to convert a word from a source to a target script on the basis of the word's pronunciation (Figure 2). Many neural transliteration approaches follow the sequence-to-sequence model originally proposed for NMT (Jadidinejad, 2016; Rosca and Breuel, 2016). Our AC-RNN transliteration system is similar to these, but with an improved objective to address exposure bias.

Other approaches have been proposed to address exposure bias in RNN, especially for NMT. Inspired by imitation learning (Vlachos, 2013), Bengio et al. (2015) introduce the notion of scheduled sampling, where the decoder RNN is gradually exposed to its own errors. While scheduled sampling exposes the system only to errors from the immediately preceding time step, reinforcement-learning techniques can consider the entire sequence. Ranzato et al. (2015) apply the REINFORCE algorithm to NMT, to train the network with a reward derived from the BLEU score of each generated sequence. Bahdanau et al. (2016) apply the Actor-Critic algorithm in NMT by applying a reward-reshaping approach to construct intermediate BLEU feedback at each step. Rennie et al. (2017) introduce a self-critical training approach that does not require a critic model, which has been successfully applied to abstract summarization (Paulus et al., 2017). Unlike these previous works that apply reinforcement-learning techniques to optimize an external metric such as ROUGE in text summarization, or BLEU in translation, giving one reward at the end of each sequence, we demonstrate that sequence labeling and transduction tasks benefit from the binary rewards that are available at each step.

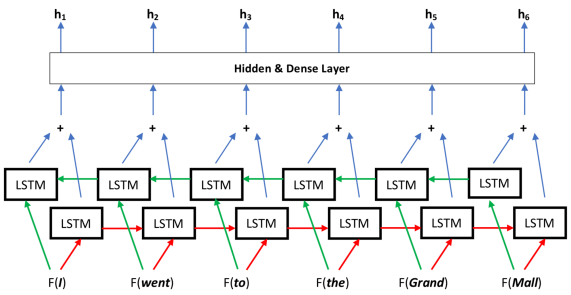

Figure 3: Time-unfolded encoder for labeling the sentence *'I went to the Grand Mall'*. F retrieves the computed feature vectors. The symbol '+' denotes the concatenation of the forward and backward outputs.

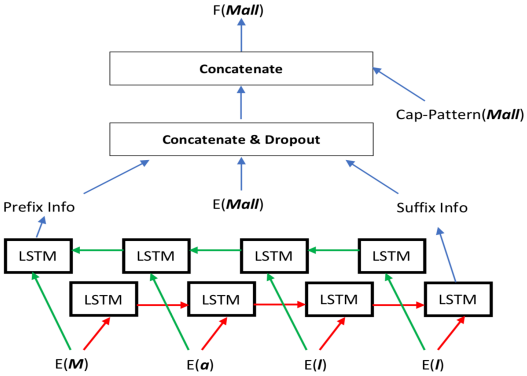

Figure 4: Encoder for building the feature vector of 'Mall'. E retrieves word or character embeddings.

3 Architecture

We are motivated by the observation that both CRFs and RNNs can be used to track the output dependencies needed to generate coherent sequences. In an analogy to encoder-decoder models, either could be used as a decoder. Following that analogy, we also need an encoder to build input features, which we share across all competing systems.

3.1 Encoder

Following Huang et al. (2015), the encoder of our sequence labeler employs a bi-directional RNN over the tokens in the sequence. The bi-directional RNN transforms context-independent token representations into representations of tokens-in-context, allowing each position to potentially encode information from the entire input sequence.

For sequence labeling, we follow Lample et al. (2016), and build our context-independent word representation by combining an embedding table with the outputs of a bi-directional RNN applied to each word's characters (Figures 3 and 4). The final states of the forward and backward character RNNs are concatenated to the word's embedding, and then passed through a dropout layer. We also concatenate capitalization pattern indicators to these feature vectors. Our word embeddings are initialized using embeddings pre-trained on a large corpus. For transliteration, where we operate exclusively on the character level, we apply a bi-directional RNN on the character representations, which are provided by a randomly initialized embedding table.

3.2 Decoders

Given an input $X = (x_1, x_2, ..., x_l)$, we look for an output sequence $Y = (y_1, y_2, ..., y_l)$ where each y_t is an output token. In the encoder (Section 3.1), we transform the input X into a sequence of hidden vectors $H = (h_1, h_2, ..., h_l)$. Given these vectors, the simplest decoder does not account for output dependencies at all. Instead, it independently predicts the output at time t by mapping h_t into a probability distribution $p_{\text{INDP}}(y_t|h_t)$ using a softmax layer. This results in a sequence-level probability of $p(Y|X) = \prod_t p_{\text{INDP}}(y_t|h_t)$.

In sequence labeling tasks, there are typically dependencies between the output tokens; for example, in English Part of Speech tagging, a determiner is unlikely to follow another determiner. The most prominent and widely used approach to track such dependencies is the CRF, which uses dynamic programming over an undirected graphical model to maintain a well-defined probability distribution over sequences, effectively modeling $p(Y|X) = p_{\text{CRF}}(Y|H)$, where p_{CRF} hides fixed-order Markov dependencies over Y. The CRF can be used as a node in a neural sequence labeler, while still allowing the system to train end-to-end (Huang et al., 2015).

An alternative technique is to use a decoder RNN on top of the encoder, as is typically done in neural machine translation (Sutskever et al., 2014), and has been done for CCG supertagging (Vaswani et al., 2016). Let d_t be the recurrent decoder state, summarizing the output sequence up to time t, and let c_t be the context vector that summarizes the input X for time t. For

Input phrase	the	University	of	XYZ
Reference tags	O	Org	Org	Org
Model's prediction	O	Org	O	Loc
J_{ml}	$\ln P(\text{O})$	$\ln P(\text{Org}\|\text{O})$	$\ln P(\text{Org}\|\text{O Org})$	$\ln P(\text{Org}\|\text{O Org Org})$
J_{ac}	$1.8 \ln P(\text{O})$	$0.9 \ln P(\text{Org}\|\text{O})$	$-0.1 \ln P(\text{O}\|\text{O Org})$	$-0.1 \ln P(\text{Loc}\|\text{O Org O})$

Table 1: The formed probabilities by maximum-likelihood (J_{ml}) and actor-critic (J_{ac}) objectives for named entity tagging of the phrase 'the University of XYZ'. The advantage terms $\delta_t = 1.8, 0.9, \ldots$ given in J_{ac} are computed using $\gamma = 0.9$, $n = 4$, and $V(t) = 0.1$.

sequence labeling, the source-to-target alignment is trivial, and c_t is provided directly by the encoder: $c_t = h_t$. We can then use the input-feeding method of Luong et al. (2015) to define d_t recursively: $d_t = \text{RNN}(d_{t-1}, y_{t-1}, c_{t-1})$, where RNN is a recurrent unit, in our case, an LSTM (Hochreiter and Schmidhuber, 1997). Finally, a softmax layer is used to define a probability distribution over output tokens $p_{\text{SM}}(y_t|c_t, d_t)$, resulting in a sequence model of: $p(Y|X) = \prod_t p_{\text{RNN}}(y_t|h_t, y_{t'<t}) = \prod_t p_{\text{SM}}(y_t|c_t, d_t)$. During training, the gold-standard previous token y_{t-1} is fed into the decoder at time t, while at test time, we use the model's generated output.

For transliteration, where the input and output sequence lengths do not match, we can no longer simply provide the encoder state h_t at time t as the context vector c_t for our probability models. Following standard practice in NMT, for the decoder RNN, an attention mechanism (Bahdanau et al., 2014) can learn an alignment model that provides a scalar score $\alpha_{t,t'}$ for each target position t and source position t', giving us a context vector $c_t = \sum_{t'} \alpha_{t,t'} h_{t'}$, which we can use in place of h_t in the RNN models described above. Specifically, we use the global-general attention mechanism of Luong et al. (2015). For the CRF, an output hidden state is not available as conditioning information for learned attention. Instead, we build context vectors $c_t = (h_{t-1}, h_t, h_{t+1})$. These concatenated vectors act as a window-based local attention (Luong et al., 2015). In order to allow the CRF to generate output of a different length from its input, we pad both sequences with extra end symbols up to a fixed maximum length, and let CRF decode until the end of the padded source sequence. It controls its target length by outputting padding tokens.

All of the models described above can be trained with a maximum likelihood objective:

$$J_{ml}(\theta) = \sum_{X,Y} \ln p_\theta(Y|X)$$

4 Biased Actor-Critic Training

To address the issue of exposure bias in our decoder RNN, we adopt the Actor-Critic (AC) algorithm, as it allows us to easily account for intermediate, tag-level rewards. In AC training, the decoder RNN first generates a greedy output sequence according to its current model, similar to how it would during testing. We calculate a sequence-level credit for each prediction by comparing it to the gold-standard. The AC update modifies our RNN to improve the expected credit at each step. This process exposes the decoder to its own errors, alleviating exposure bias. Algorithm 2 provides pseudo code for the training process, which we expand upon in the following paragraphs.

Inspired by Maes et al. (2009), we define the token-level reward R_t as $+1$ if the generated token $\hat{y}_t$ is the same as the gold token y_t, and as 0 otherwise (Hamming loss). We compute the sequence-level credit $S(t)$ for each decoding step using the multi-step Temporal Difference return (Sutton and Barto, 1998):

$$S(t) = \sum_{i=0}^{n-1} [R_{t+i}] + V_{\theta'}(t+n)$$

The step count n allows us to control our bias-variance trade-off, with a large n resulting in less bias but higher variance. The critic $V_{\theta'}(t)$ is a regression model that estimates the expected return $E[S(t)]$, taking the context vector c_t and the decoder's hidden state d_t as input. It is trained jointly alongside our decoder RNN, using a distinct optimizer. With this critic in place, the update for the AC algorithm is defined as

$$\frac{\partial J_{ac}(\theta)}{\partial \theta} = \sum_t \frac{\partial \ln(p_\theta(\hat{y}_t|X, \hat{y}_{t'<t}))}{\partial \theta} (\delta_t)$$

where:

$$\delta_t = S(t) - V_{\theta'}(t)$$

The AC update changes the prediction likelihood proportionally to the advantage δ_t of the token $\hat{y}_t$. Therefore, if $S(t) > V_{\theta'}(t)$, the decoder should increase the likelihood. The AC error $\frac{\partial J_{ac}(\theta)}{\partial \theta}$

Algorithm 2 Biased Actor-Critic Training

- Given: n as hyper-parameter
- Input: Source X and Target Y
- Greedy decode X using θ to get:
  - the output sequence $\hat{Y} = (\hat{y}_1, ..., \hat{y}_{l'})$
  - decoder RNN states $D = (d_1, ..., d_{l'})$
  - context vectors $C = (c_1, ..., c_{l'})$
- For each output target position t:
  - $r_t = 1$ if $\hat{y}_t = y_t$, 0 otherwise
  - $V_{\theta'}(t) = \text{ValueNetwork}(d_t, c_t, \theta')$
- $loss_\theta = 0$; $loss_{\theta'} = 0$
- For each output target position t:
  - $S(t) = \sum_{i=0}^{n-1} [r_{t+i}] + V_{\theta'}(t+n)$
  - $\delta_t = S(t) - V_{\theta'}(t)$
  - $b\delta_t = \text{bias}(y_t, \hat{y}_t, \delta_t) \times \delta_t$
  - $loss_\theta = loss_\theta - b\delta_t \ln p_\theta(\hat{y}_t | X, \hat{y}_{t'<t})$
  - $loss_{\theta'} = loss_{\theta'} + \delta_t \times \delta_t$
- Back-propagate through $loss_\theta$ and $loss_{\theta'}$ as normal to update θ and θ' respectively
- Do not Back-propagate into $S(t)$ to update θ'

back-propagates only through the actor's prediction likelihood p_θ.

Table 1 illustrates an example in NER, where the model tends to incorrectly label an entity as 'Location' instead of 'Organization'. We directly give negative credits for the invalid predictions with the J_{ac} objective.

Critic Architecture: We employ a non-linear feed-forward neural network as our critic, which uses leaky-ReLU activation functions (Nair and Hinton, 2010) in the first two hidden layers. In the output layer, it uses a linear transformation to generate a scalar value. The ValueNetwork in Algorithm 2 refers to this critic model. To learn the parameter set θ' in the critic, we use a semi-gradient update. We do not use the full gradient in the Mean Squared error to train this regression model. Accordingly, for $\frac{\partial loss_{\theta'}}{\partial \theta'} = \frac{\partial(\delta_t \times \delta_t)}{\partial \theta'}$, instead of using $2\delta_t \frac{\partial(S(t) - V_{\theta'}(t))}{\partial \theta'}$, we use the update $2\delta_t \frac{\partial V_{\theta'}(t)}{\partial \theta'}$. The temporal difference return $S(t)$ uses the critic's estimate $V_{\theta'}(t+n)$. The full gradient will cause a dangerous feedback loop as by doing so, we will allow $S(t)$ to match

Figure 5: The effect of biased training in Actor-Critic objective on CCG supertagging (top) and German NER (bottom).

with $V_{\theta'}(t)$ in order to reduce the Mean Squared error, however, in the critic model, we need to improve our previous estimates to match with the newly observed $S(t)$ (Sutton and Barto, 1998).

Biased Training: Due to the inevitable regression error of the critic, and the fact that it is randomly initialized at the beginning, the advantage δ_t can undesirably become negative for a correctly-selected tag, or positive for a wrongly-selected tag. Optimizing the network according to these invalid advantages would increase the probability of the wrong tags, while decreasing the probability of the true tag. In such cases, to help critic update itself and form better estimates in the next iteration, we clip δ_t to zero by defining the biased advantage $b\delta_t$ as $\text{bias}(y_t, \hat{y}_t, \delta_t) \times \delta_t$ where:

$$\text{bias}(y_t, \hat{y}_t, \delta_t) = \begin{cases} 0 & \text{if } \hat{y}_t = y_t \ \& \ \delta_t < 0 \\ 0 & \text{if } \hat{y}_t \neq y_t \ \& \ \delta_t > 0 \\ 1 & \text{otherwise} \end{cases}$$

By setting the advantage $b\delta_t$ to 0, the bias term effectively switches off the entire actor update when the advantage has the wrong polarity. This introduces bias, but also serves to stabilize training. Note that the critic is always updated.

For the overall training of the encoder-decoder RNN, we pre-train the network using J_{ml}. We then continue training from the best model using J_{ac}. Figure 5 shows development set performance, starting from the same J_{ml}-pre-trained point, for our biased objective, as compared to standard Actor-Critic on CCG supertagging and German NER. We observe that the biased training is crucial for Actor-Critic objective to improve

Hyper-parameter	En/De	CCG	TL
char_embed_size	32	32	128
output_embed_size	32	128	128
max_gradient_norm	5.0	5.0	10.0
encoder units	256	512	256
decoder units	256	512	256
batch size	32	10	64
n	2/4	8	6
dropout	0.5	0.5	0.5
RNN gate	LSTM	LSTM	LSTM

Table 2: Hyper-parameters used in the experiments.

upon the pre-trained maximum likelihood model on CCG supertagging, and helps the model reach a higher point on NER.

5 Experiments

We compare AC-RNN, CRF, and RNN on three tasks: NER tagging, CCG supertagging, and transliteration. We then compare the biased Actor-Critic objective with Scheduled Sampling on NER. Finally, we compare biased AC training to the Self-Critical method of Rennie et al. (2017) on transliteration.

5.1 Setup

Datasets: To conduct the NER experiments, we use the English and German datasets of the CoNLL-2003 shared task (Tjong Kim Sang and De Meulder, 2003), and employ the 'BILOU' tagging scheme[2] (Ratinov and Roth, 2009). For CCG supertagging, we use the English CCGbank (Hockenmaier and Steedman, 2007), the standard sections {02-21}, {00}, and {23} as the train, development, and test sets, respectively. We use pre-trained, 100-dimensional *Glove* embeddings for all English word-level tasks (Pennington et al., 2014), and fine-tune them during training. For German NER, we obtain the embeddings (64 dimensions) of Lample et al. (2016), which are trained on a German monolingual dataset from the 2010 Machine Translation Workshop. We apply no preprocessing on the datasets except replacing the numbers and unknown words with the 'NUM' and 'UNK' symbols.

We conduct the transliteration experiments on the English-to-Chinese (EnCh), English-to-Japanese (EnJa), English-to-Persian (EnPe), and English-to-Thai (EnTh) datasets of the NEWS-2018 shared task.[3] The training sets contain approximately 40K, 30K, 10K and 30K instances for EnCh, EnJa, EnPe, and EnTh, respectively, while the development sets have 1K instances. We train the models on the training sets, and evaluate them on the development sets. We hold out 10% of the training sets as our internal tuning sets.

Model	Dev	Test
INDP	93.63 ±0.13	89.77 ±0.21
RNN	94.43 ±0.16	90.75 ±0.23
CRF	94.47 ±0.12	90.80 ±0.19
AC-RNN	**94.54** ±0.12	**90.96** ±0.15
Lample et al. (2016)		90.94

Table 3: Average entity-level F1-score for English NER on the CoNLL-2003 datasets. Reimers and Gurevych (2017) report 90.81 as the median performance for the CRF model of Lample et al. (2016) trained with 41 randoms seeds.

Model	Dev	Test
INDP	75.51 ±0.28	72.15 ±0.57
RNN	76.85 ±0.39	73.52 ±0.36
CRF	76.27 ±0.35	73.59 ±0.36
AC-RNN	**77.10** ±0.29	**73.82** ±0.29
Lample et al. (2016)		78.76

Table 4: Average entity-level F1-score for German NER on the CoNLL-2003 dataset.

Training: We implement all our models in a shared code base, and all models share the same encoder, using the same number of hidden units. Table 2 summarizes the hyper-parameters used in our experiments. The maximum likelihood training is done with the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 0.0005. The AC training is done with the mini-batch Gradient Descent ($\theta = \theta + \alpha \frac{\partial J_{ac}(\theta)}{\partial \theta}$) using a fixed step size of 0.5 for NER & CCG, and 0.1 for transliteration experiments. The critic is trained with a separate Adam optimizer with the learning rate of 0.0005. We employ a linear-chain first-order undirected graph in the CRF models.

As performance varies depending on the random initialization, we train each model 20 times for NER and 5 times for CCG using different random seeds which are the same for all models. We report scores averaged across these runs ±

[2] Also known as the 'IOBES' tagging scheme.

[3] http://workshop.colips.org/news2018/shared.html

Model	Dev	Test
INDP	94.24 ±0.03	94.25 ±0.11
RNN	94.25 ±0.06	94.28 ±0.09
CRF	94.31 ±0.07	94.15 ±0.11
AC-RNN	**94.43** ±0.08	**94.39** ±0.06
Vaswani et al. (2016)	94.24	94.50
Kadari et al. (2017)	94.37$^\diamond$	94.49$^\diamond$

Table 5: Average top-1 accuracy for English CCG supertagging. The $^\diamond$ results are achieved with CRF training.

Model	Memory (GB)	Time (m)
INDP	**1.3**	**5**
RNN	1.8	11
CRF	9.8	50
AC-RNN	1.5	10

Table 6: The required GPU memory and execution time for one training epoch of the models on CCG supertagging using mini-batches of size 10.

the standard deviations. Due to time constraints, for the transliteration experiments, we train each model only once.

Evaluation: We compute the standard evaluation metric for each task: entity-level F1-score for NER, tagging accuracy for CCG, and word-level accuracy for transliteration. For the models with decoder RNNs, we report the results with a beam of size 10. For the NER and CCG experiments, we conduct the significance tests on the unseen final test sets, using the Student's t-test over random replications at the significance level of 0.05.

5.2 Results

Main Comparisons: As our primary empirical study, we compare the AC-RNN to CRF and RNN models. For NER and CCG experiments, we consider independent prediction of the labels (INDP, see Section 2) as another baseline.

The results of the NER experiments are shown in Tables 3 and 4. We observe that by modeling the output dependencies in the RNN, we achieve a significant improvement over the baseline INDP, about 1% F1-score on both English and German datasets. With respect to prior work, our CRF model replicates the reported results on English NER.[4] Moreover, AC-RNN significantly outperforms both RNN and CRF on both English and German test sets. These results demonstrate that AC-RNN is successful at overcoming the RNN's exposure bias, and represents a strong alternative to CRF for named entity recognition.

[4]On German NER, we cannot replicate the CRF results of Lample et al. (2016), although we obtained their German word embeddings. We attribute this discrepancy to different preprocessings of the dataset.

Model	EnCh	EnJa	EnPe	EnTh
CRF	67.6	45.8	75.6	32.2
RNN	70.6	51.6	76.3	39.7
AC-RNN	**72.3**	**52.4**	**77.8**	**41.4**
OpenNMT	70.1	47.7	70.5	36.3

Table 7: The word-level transliteration accuracy on the development sets of NEWS-2018 shared task.

On CCG supertagging (Table 5), AC-RNN is significantly better than all other models, and is competitive with reported state-of-the-art results. For this task, we had expected the improvements to be larger, because of CCG supertagging's potential for long-distance output dependencies. Instead, the results show that independent predictions do surprisingly well.

The CCG experiments reveal that AC-RNN is trained more efficiently than CRF. Table 6 shows the time and memory requirements for each method. We observe that, due to the large output vocabulary size of the task (1284 supertags), CRF is five times slower than AC-RNN during training, while the batched version of its forward algorithm requires six times more GPU memory.[5]

The transliteration results in Table 7 show that AC-RNN outperforms CRF (likely due to CRF's local attention), as well as RNN (likely due to exposure bias). The transliteration experiments support our hypothesis that AC-RNN is more generally-applicable than CRF, and the improvements from AC training transfer to other sequence-to-sequence tasks.

To confirm that our RNN baseline performs reasonably well, we validate our transliteration model against a standard NMT implementation as provided by the OpenNMT tool (Klein et al., 2017). We apply the tool "as-is" with its default translation hyper-parameters. Note that our RNN system in this experiment is also an NMT-style sequence-to-sequence model with attention.

[5]The forward algorithm of CRF has a memory complexity of θ(batch size $\times$ #tags $\times$ #tags), and runs out of memory with the mini-batch size of 16 on a 12-GB Graphical Processing Unit.

Model	Dev	Test
RNN	76.85 ±0.39	73.52 ±0.36
SS-RNN	76.93 ±0.32	73.65 ±0.29
AC-RNN	**77.10** ±0.29	**73.82** ±0.29

Table 8: The biased Actor-Critic training compared to Scheduled Sampling on German NER.

Model	EnCh	EnJa	EnPe	EnTh
RNN	70.6	51.6	76.3	39.7
SC-RNN	70.2	51.8	77.2	41.3
AC-RNN	**72.3**	**52.4**	**77.8**	**41.4**

Table 9: The biased Actor-Critic objective compared to the Self-Critical policy training on the development sets of the NEWS-2018 shared task.

Scheduled Sampling Comparison: In the next experiment, we compare the biased AC objective to scheduled sampling, which also addresses exposure bias in RNNs. In this approach (Bengio et al., 2015), at each time step t with probability ϵ, we feed the gold-standard token y_{t-1} into the decoder RNN, otherwise we use the model's greedily-generated token $\hat{y}_{t-1}$. The sampling probability is annealed at every training epoch so that we use gold-standard inputs at the beginning of the training, but while approaching the end, we instead condition the predictions on the model-generated inputs. We implement this approach using the inverse sigmoid schedule of Bengio et al. (2015) for annealing ϵ, and denote it as SS-RNN.

Table 8 shows the comparison of SS-RNN with AC-RNN on German NER. Both systems improve over RNN, but AC-RNN is significantly better than SS-RNN. This result supports our hypothesis that the reinforcement-learning solutions should outperform the scheduled sampling, as AC training considers the entire sequence, whereas the scheduled sampling addresses only exposure to the immediately previous token. We also observe that the scheduled sampling, unlike biased AC training, is highly sensitive to the choice of sampling schedule.

Self-Critical Comparison: In our final experiment, we compare the biased AC training with the Self-Critical policy training of Rennie et al. (2017) which does not require a critic model. This method is intended to represent the state-of-the-art in reinforcement learning for sequence-to-sequence models with sequence-level rewards, to be contrasted against our AC-RNN and its position-level rewards.

The SC training samples an action based on the probability distribution formed over all actions at each step, resulting in the sampled sequence $s\hat{Y} = (s\hat{y}_1, ..., s\hat{y}_{l'})$. The advantage for $s\hat{Y}$ is defined compared to the greedily-sampled sequence $\hat{Y} = (\hat{y}_1, ..., \hat{y}_{l'})$. We compare AC-RNN with SC-RNN on transliteration, where SC-RNN's sequence-level errors are the edit distances between the sampled sequences and the gold target Y, optimizing the objective $J_{sc} = \sum_{X,Y} \Big(\Delta(\hat{Y}, Y) - \Delta(s\hat{Y}, Y)\Big) \times \log P_\theta(s\hat{Y}|X)$. The rest of the training details are kept the same for both objectives, except that we cannot apply biased training to the SC objective. If the sampled sequence $s\hat{Y}$ has a lower edit distance compared to the greedily-sampled sequence $\hat{Y}$, the likelihood of $s\hat{Y}$ is increased by the policy P_θ.

The results shown in Table 9 indicate that the Self-Critical training improves over RNN on EnJa and EnPe, and EnTh, however, it fails to beat AC-RNN across all the evaluation sets. This observation is aligned with our initial hypothesis that the reinforcement-learning techniques, applied to sequence labeling and transduction, would benefit more from modeling the intermediate rewards, as is done with the Temporal Difference credits in AC training (Section 4).

6 Conclusion

We have proposed a biased Actor-Critic algorithm to train encoder-decoder RNNs for sequence labeling and transduction. Though Actor-Critic and related reinforcement-learning algorithms have previously been applied to sequence-to-sequence learning, our proposed AC-RNN is specialized to sequence-labeling by taking advantage of the per-position rewards. To our knowledge, we have presented the first direct, controlled comparison between CRFs and any form of RNN. On NER and CCG supertagging, our system significantly outperforms both RNN and CRF, establishing the AC-RNN as an efficient alternative for sequence labeling. The advantages of AC-RNN in terms of efficiency and flexibility include fast training, small memory footprint, and the ease of application to other transduction tasks, such as transliteration. Finally, we showed that our proposal for

handling exposure-bias outperforms the related alternatives of scheduled sampling and self-critical policy training.

References

Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron C. Courville, and Yoshua Bengio. 2016. An actor-critic algorithm for sequence prediction. *CoRR*, abs/1607.07086.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems 28*, pages 1171–1179. Curran Associates, Inc.

Stephen Clark and James R. Curran. 2004. The importance of supertagging for wide-coverage ccg parsing. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*.

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

Julia Hockenmaier and Mark Steedman. 2007. Ccgbank: A corpus of ccg derivations and dependency structures extracted from the penn treebank. *Comput. Linguist.*, 33(3):355–396.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF models for sequence tagging. *CoRR*, abs/1508.01991.

Amir Hossein Jadidinejad. 2016. Neural machine transliteration: Preliminary results. *CoRR*, abs/1609.04253.

Rekia Kadari, Yu Zhang, Weinan Zhang, and Ting Liu. 2017. Ccg supertagging via bidirectional lstm-crf neural architecture. *Neurocomputing*.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush. 2017. OpenNMT: Open-Source Toolkit for Neural Machine Translation. *ArXiv e-prints*.

Vijay R Konda and John N Tsitsiklis. 2003. On actor-critic algorithms. *SIAM journal on Control and Optimization*, 42(4):1143–1166.

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

Mike Lewis, Kenton Lee, and Luke Zettlemoyer. 2016. Lstm ccg parsing. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 221–231. Association for Computational Linguistics.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421. Association for Computational Linguistics.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1064–1074. Association for Computational Linguistics.

Francis Maes, Ludovic Denoyer, and Patrick Gallinari. 2009. Structured prediction with reinforcement learning. *Mach. Learn.*, 77(2-3):271–301.

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA. Omnipress.

Romain Paulus, Caiming Xiong, and Richard Socher. 2017. A deep reinforced model for abstractive summarization. *CoRR*, abs/1705.04304.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2015. Sequence level training with recurrent neural networks. *CoRR*, abs/1511.06732.

Lev Ratinov and Dan Roth. 2009. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL-2009)*, pages 147–155, Boulder, Colorado. Association for Computational Linguistics.

Nils Reimers and Iryna Gurevych. 2017. Reporting score distributions makes a difference: Performance study of lstm-networks for sequence tagging. In *Proceedings of the 2017 Conference on Empirical*

Methods in Natural Language Processing, pages 338–348, Copenhagen, Denmark. Association for Computational Linguistics.

Steven J. Rennie, Etienne Marcheret, Youssef Mroueh, Jarret Ross, and Vaibhava Goel. 2017. Self-critical sequence training for image captioning. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1179–1195.

Mihaela Rosca and Thomas Breuel. 2016. Sequence-to-sequence neural network models for transliteration. *CoRR*, abs/1610.09565.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27*, pages 3104–3112.

Richard S. Sutton and Andrew G. Barto. 1998. *Introduction to Reinforcement Learning*, 1st edition. MIT Press, Cambridge, MA, USA.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL-2003*, pages 142–147. Edmonton, Canada.

Ashish Vaswani, Yonatan Bisk, Kenji Sagae, and Ryan Musa. 2016. Supertagging with lstms. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 232–237. Association for Computational Linguistics.

Andreas Vlachos. 2013. An investigation of imitation learning algorithms for structured prediction. In *Proceedings of the Tenth European Workshop on Reinforcement Learning*, volume 24 of *Proceedings of Machine Learning Research*, pages 143–154, Edinburgh, Scotland. PMLR.

Huijia Wu, Jiajun Zhang, and Chengqing Zong. 2017. A dynamic window neural network for ccg supertagging. In *AAAI Conference on Artificial Intelligence*.

Wenduan Xu. 2016. Lstm shift-reduce ccg parsing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1754–1764. Association for Computational Linguistics.