Sequence Labeling and Transduction with Output-Adjusted Actor-Critic Training of RNNs

by

Saeed Najafi

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

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 with the maximum-likelihood training. We frame the prediction of the output sequence as a sequential decision-making process, where the RNN takes a series of actions without being conditioned on the ground-truth labels. We then train the network with an output-adjusted actor-critic algorithm (AC-RNN). We comprehensively compare this strategy 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 show that the output-adjusted actor-critic training is significantly better than other techniques for addressing RNN's exposure bias, such as Scheduled Sampling, and Self-Critical policy training.

Preface

The work presented in this dissertation is the elaboration of a research article advised jointly by Professors Greg Kondrak and Colin Cherry (Najafi et al., 2018a). The author was the main contributor who implemented different methods, and conducted all experiments. The developed methods in this dissertation have been used in similar NLP tasks (Nicolai et al., 2018; Najafi et al., 2018c,b).

Acknowledgements

I was very lucky to have two great supervisors; Thank you Greg and Colin. I was a naive graduate student whom you helped to do research.

I must appreciate our great NLP group for all those exciting shared tasks we participated. Thank you Garrett, Bradley, Mohammad, Rashed, and Leyuan. I should not also forget my great teaching supervisors Joerg, Davood, Denilson, and Mario. I really enjoyed the department; hunting all those free pizza at AI seminars, and cookies in RL tea-time talks.

Looking 10 years back, I see the great support of my parents, and my brother everywhere. Without you, I could not make it.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The language can be viewed as a sequence of written or spoken tokens. An English sentence is a left-to-right sequence of words/punctuations. Similarly, words in Persian are right-to-left sequence of characters. Accordingly, in many of the Natural Language Processing (NLP) tasks, we predict an output sequence $Y = (y_1, y_2, y_3, ..., y_{l'})$ given the input sequence $X = (x_1, x_2, x_3, ..., x_l)$, where there is a structure among both input and output tokens. As an example, for translating an English sentence into German, we should generate an output sequence of German words, which convey the same English side meaning following German grammar. This structured prediction problem also arises in other domains such as image segmentation in computer vision (Nowozin and Lampert, 2011) or Gene prediction in bioinformatics (Culotta et al., 2005). In NLP, we identify three versions for this problem:

- **Sequence Labeling**:

  The lengths of input and output sequences are the same ($l = l'$), and there is a one-to-one monotonic mapping between each x_i and y_i. In this version, we label each token x_i with one of the possible output tags. As an example, in the Named Entity Recognition (NER) task, for the input sentence 'I study at the University of Alberta', we assign one predefined type ('Organization', 'Person', etc.) to each input token. The multi-word entity 'University of Alberta' is tagged as an organization resulting in the output sequence '- - - - Org Org Org'.

  We can also consider the task of Combinatory Categorical Grammar

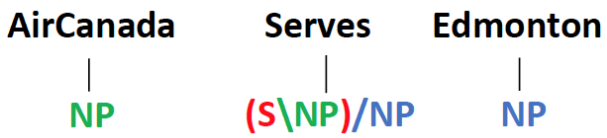

Figure 1.1: CCG supertagging of the sentence *'AirCanada Serves Edmonton'*.

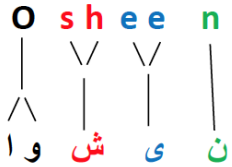

Figure 1.2: Transliteration of the name *'Osheen'* into Persian.

(CCG) supertagging (Clark and Curran, 2004), where we label each word in a sentence with one of 1,284 morphosyntactic categories from CCGbank (Hockenmaier and Steedman, 2007). The example shown in Figure 1.1 illustrates that the given supertag to the verb 'Serves' contains this information that the verb is preceded by one noun phrase on the left (denoted by backward slash), and is followed by another noun phrase on the right (denoted by forward slash).

- **Sequence Transduction**:

  The lengths of input and output sequences might not match ($l \neq l'$), and there are many-to-many mappings between input and output tokens, still these mappings are monotonically aligned. As a character-level transduction task, we can consider transliteration where the goal is to convert a word from a source to a target script on the basis of the word's pronunciation (Knight and Graehl, 1998). The Figure 1.2 illustrates the alignments for the transliteration of the name 'Osheen' into Persian where there are 1-2, 2-1, and 1-1 mappings between the English and Persian letters.

- **General Sequence-to-Sequence**:

  In a general case, the translation task has input and output sequences of different lengths, and there are non-monotonic alignments (with re-

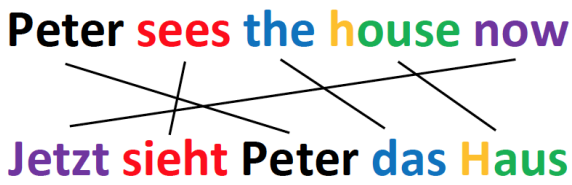

Figure 1.3: Translation of the sentence *'Peter sees the house now'* into German.

ordering) between the source and target tokens (Figure 1.3). An English sentence with the grammar rule 'subject + verb + object + adverb' will not necessarily keep the same order of its parts-of-speeches once translated into a different language (e.g. 'adverb + verb + subject + object' in German).

Throughout this work, we only focus on sequence labeling and transduction tasks. We investigate statistical methods for modelling the discriminative distribution $P(Y|X)$. Given an input X, during the inference phase, we search for the optimal output Y that maximizes $P(Y|X)$.

In order to predict the output sequence Y given the input X, we should capture two types of dependencies. First, we need to capture the context-dependent relations among input tokens. This goal was previously achieved with hand-engineered features defined for a bag of words or characters appeared together in the sentence. Recently, deep neural architectures are dominant for this purpose, especially Recurrent Neural Networks (RNN) (Goodfellow et al., 2016) which have been widely used as the feature representation layer across all NLP tasks, including NER (Huang et al., 2015), CCG supertagging (Kadari et al., 2017; Wu et al., 2017; Lewis et al., 2016; Xu, 2016; Vaswani et al., 2016), transliteration (Jadidinejad, 2016; Rosca and Breuel, 2016), and machine translation (Sutskever et al., 2014).

The second type of dependencies is among the output tokens. The Conditional Random Field (CRF) (Lafferty et al., 2001) is a popular probabilistic model to capture these output dependencies. It has been widely used in sequence labeling models with hand-engineered features (Finkel et al., 2005;

Ratinov and Roth, 2009), as well as in recent models with neural feature representations (Huang et al., 2015; Chiu and Nichols, 2016; Lample et al., 2016). The CRF has been less effective in transduction and translation tasks mainly due to its inability to generate an output sequence of a different length from its input, and its costly computation in tasks with large output vocabularies. Accordingly, recent transliteration and translation models employ another RNN over the output sequence to capture output dependencies (Grundkiewicz and Heafield, 2018; Jadidinejad, 2016; Rosca and Breuel, 2016; Luong et al., 2015; Sutskever et al., 2014). This RNN is referred to as a decoder RNN, and the RNN-based representation layer is referred to as encoder. The encoder-decoder RNNs have been used in various NLP tasks, such as grammatical error correction (Yuan and Briscoe, 2016), sentence summarization (Rennie et al., 2017) or dialogue generation (Li et al., 2017). Still, all recent works on sequence labeling use CRF to model output dependencies (Moon et al., 2018; Xu et al., 2017; Strubell et al., 2017; Yang et al., 2016; Bharadwaj et al., 2016; Kuru et al., 2016; Gillick et al., 2016).

1.1 Motivation

A general approach is needed to capture output dependencies for various structured prediction tasks in NLP. In this dissertation, we set out to establish the decoder RNN as an attractive alternative to the CRF in order to provide such a general method. 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. Unifying methods for different NLP tasks would simplify multitask learning, which is necessary for real NLP-oriented applications (e.g. information retrieval).

1.2 Challenges

The CRF is an undirected Random field which defines a sequence-level score for each possible output sequence. It then globally normalizes these scores to form the distribution $P(Y|X)$ (details will be discussed in section 3.3). Al-

ternatively, the decoder RNN (section 3.4) is a left-to-right directed Bayesian network, which is trained with token-level locally-normalized objectives. 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; Ranzato et al., 2016; Wiseman and Rush, 2016). RNNs are typically conditioned on ground-truth labels 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.

1.3 Objectives

In this work, we will propose an effective training strategy to counter RNN's exposure-bias problem. To do so, we will frame the prediction of the output sequence as a sequential decision-making process, where the decoder RNN takes a series of actions without being conditioned on the ground-truth labels. We will employ an actor-critic reinforcement learning objective (Konda and Tsitsiklis, 2003; Mnih et al., 2016) to train the model using the binary rewards defined for each action. We will introduce the supervision of the gold labels into this objective through an output-adjusted training. Overall, the new training procedure will completely expose the RNN to its own errors.

We will conduct comprehensive analysis comparing against CRF under controlled conditions using a shared implementation. We will experiment on three structured output tasks: NER, CCG supertagging, and transliteration. Moreover, we will compare our proposed training strategy with previous methods suggested for addressing exposure bias. We will empirically demonstrate that our training method is significantly better than Scheduled Sampling (Bengio et al., 2015). We will also demonstrate that our approach is more suitable for sequence labeling and transduction tasks than other reinforcement learning methods such as Self-Critical method of Rennie et al. (2017).

1.4 Outline

This work is organized as follows. We first discuss the preliminary background information on Language Models and Recurrent Neural Networks in Chapter 2. In Chapter 3, we investigate the current state-of-the-art feature representation layers for each task, and discuss CRF and decoder RNNs in detail. The Chapter 4 introduces our output-adjusted actor-critic algorithm. Our comprehensive experiments are presented in Chapter 5. We then provide future directions in Chapter 6, and summarize this dissertation in Chapter 7.

Chapter 2

Sequence Modelling

In this chapter, we provide the background information needed to design our discriminative models to compute $P(Y|X)$. We first start with the concept of Language Modelling (LM), which assigns a probability to a sentence representing how natural that sentence is in a specific language. We briefly discuss n-gram, and feed-forward neural LMs. We then review recent techniques for mapping words and characters into real-valued feature vectors. Finally, we discuss recurrent neural LMs.

2.1 N-gram and Feed-Forward LM

There is no doubt that language evolves over time. Thousands of different languages are spoken around the world. It is infeasible to model a language by developing rules for every usage, and updating them for every newly emerged phrase. What does make 'Sup, yo' a grammatical English phrase compared to 'Soup, yo'? The former can be supported by collecting thousand dialogues, whereas the latter is a bizarre usage of the food 'soup' with the slang 'yo' (hello).

The statistical language modelling aims at assigning the probability $P(X)$ to the sentence X, which represents how natural the sentence X is in a specific language. If we collect 100 English phrases where we observe the phrase 'Sup, yo' once, and 'What's up' 5 times, we can estimate P('Sup, yo')=0.01 and P('What's up')=0.05. The phrase 'Soup, yo' gets a zero probability as it never appears in our corpus. Still, we cannot learn the distribution P

for every combination through counting. Let assume X consists of l tokens, $X = (x_1, x_2, x_3, ..., x_l)$, and we have 100,000 types (i.e. distinct tokens) in total. There would be $100,000^l$ combinations of these types. The unseen combinations cannot always have a zero probability. The phrase 'Yo, what's up?' never appears in our collected corpus, but it is grammatically correct.

The n-gram LMs (Jurafsky and Martin, 2000) learn the distribution P by applying the following assumptions:

- Employing the multiplication rule to learn the probability of each token conditioned on its left context:
  $P(X = x_1, x_2, x_3, ..., x_l) \doteq P(x_1) \times P(x_2|x_1) \times P(x_3|x_1, x_2) \times ... \times P(x_l|x_1, ..., x_{l-1})$
- Considering only the previous $n-1$ tokens:
  $P(x_t|x_1, ..., x_{t-1}) \approx P(x_t|x_{t-n+1}, ..., x_{t-1})$

By counting x_{t-n+1}^{t} and x_{t-n+1}^{t-1} in a corpus, $P(x_t|x_{t-n+1}^{t-1})$ is estimated with $\frac{\text{Count}(x_{t-n+1}^{t})}{\text{Count}(x_{t-n+1}^{t-1})}$. In practice, $n < 7$ as many of the high-order n-grams are rarely present in any corpora. The counting approach treats two semantically-similar types (e.g. Queen & King) as completely different concepts. It is desirable to generalize from the phrases 'Queen was in charge' and 'The King is responsible' to the sentence 'The Queen was responsible'. The n-gram LMs rely on smoothing techniques (Jurafsky and Martin, 2000) to assign a non-zero probability to the unseen bigram 'was responsible'.

In a seminal work, Bengio et al. (2003) represent each token as a real-valued vector in R^n, and jointly learn these vectors along with the distribution P using a feed-forward neural network, which could successfully surpass n-gram LMs. As an example, the proposed network predicts the token 'responsible' conditioned on the trigram 'The King is':

$$\begin{aligned} f &= concat\{em(\text{the}), em(\text{king}), em(\text{is})\} \\ h &= tanh(W \times f + b_1) \\ s &= U \times h + b_2 \\ p(\text{responsible}|\text{the, king, is}) &= \frac{e^{s_{\text{responsible}}}}{\sum_{\text{word}} e^{s_{\text{word}}}} \end{aligned} \tag{2.1}$$

The input of the network is the concatenated vectors of the tokens 'the', 'king', and 'is' extracted from a word-embedding matrix M (we assume the

function em looks up the corresponding column in the embedding matrix M). To train this network, the probability of 'responsible' given its left context is maximized through back-propagating the cross entropy objective to the parameters W, U, b_1, b_2, and the embedding matrix M (the number of labels is equal to the vocabulary size). The vectors for two semantically-similar concepts can now potentially be close in R^n, and despite n-gram LMs, the neural LM can generalize to other similar concepts (i.e. 'Queen') trained only with the previously seen concept 'King' (Bengio et al., 2003). Still, the n-gram and feed-forward LMs cannot capture long-range dependencies in a sentence or paragraph since both are limited to short window-based contexts ($n < 7$).

2.2 Embeddings

The word embeddings are created by mapping words into limited-in-dimension vectors. *Word2vec*[1], introduced by Mikolov et al. (2013), uses a feed-forward neural network to learn these vectors. In the Skip-Gram architecture of *word2vec*, the network is trained on a large corpus in order to predict the surrounding words of each token. In the sentence 'The King is responsible' assuming the token 'King' is the centre word, and we have a context window of size 2, the Skip-Gram is trained according to the following feed-forward network using the training instances ('King', 'the'), ('King', 'is'), and ('King', 'responsible'):

$$\begin{aligned} f &= em(\text{king}) \\ s &= U \times f + b_1 \\ p(\text{responsible}|king) &= \frac{e^{s_{\text{responsible}}}}{\sum_{\text{word}} e^{s_{\text{word}}}} \end{aligned} \tag{2.2}$$

In the network 2.2, d is the length of the input word vector f, and the output matrix U has v rows, and d columns, where v is the vocabulary size. If the word j appears in the context of the input word i, by maximizing the softmax function above, we maximize the dot product of the vectors $em(i)$ and $U_{j,\cdot}$. At the end of training, two words, which co-occur frequently in the language, will be close points in the vector space.

[1] https://code.google.com/archive/p/word2vec/

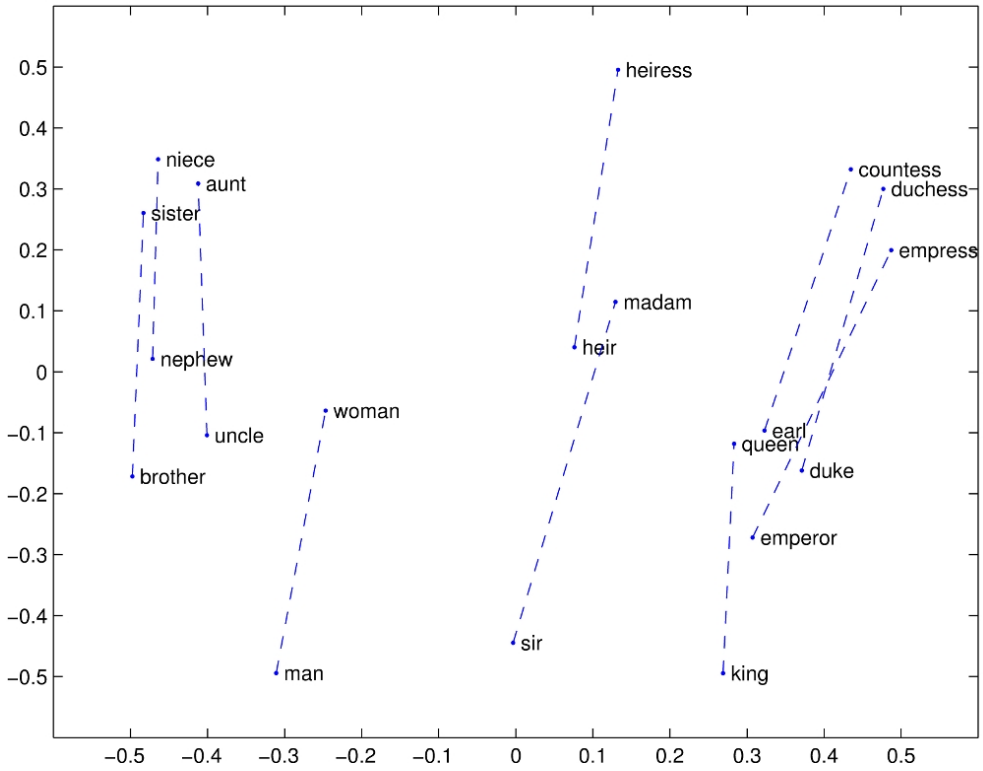

Figure 2.1: The semantic relations captured with *Glove* word embeddings for concepts around man-woman.

The advent of Graphical Processing Units (GPU) for fast training neural networks on billion words has paved the way for the derived word embeddings to be a helpful prior knowledge for NLP tasks. Similarly, Pennington et al. (2014) introduce *Glove*[2] embeddings which uses a log-bilinear regression model combining the context prediction approach of *word2vec* with co-occurrence statistics of two arbitrary words. The *Glove* is trained with the objective $J = \sum_{i,j=1}^{v} g(F_{ij})(em(i) \cdot U_{j,} + b_i + b'_j - logF_{ij})$, where F_{ij} is the co-occurrence frequency of the words j and i, b_i and b'_j are the corresponding scaler biases, and g is an empirically designed function smoothing too rare or too frequent co-occurrences. The Figure 2.1 illustrates the captured semantic relations for types related to man-woman. We can simply re-construct the vector of 'Queen' from the vectors of 'Man', 'Woman', and 'King'. In this work, we will use these pre-trained *Glove* embeddings in our experiments.

Several NLP tasks are operating on low-level types such as on charac-

[2]https://nlp.stanford.edu/projects/glove/

ters (e.g. 's' and 'h') in transliteration, or on phonemes (e.g. /ʃ/ sound of 'sh') in speech recognition. The aforementioned approaches for learning word embeddings can also be applied to build character embeddings, however, it is sufficient to initialize character vectors randomly, and jointly learn them within the NLP task as many of the alphabets have less than hundred letters.

It is still arguable that a single vector representation for each type cannot model the full semantic and grammatical regularities of language. There is ongoing research to include morphology information (e.g. 'writing' compared to 'wrote') into word embeddings (Nicolai et al., 2015; Bojanowski et al., 2017), and to differentiate embeddings of polysemous words (e.g. bank as the money institution compared to the bank of river) (Arora et al., 2016).

2.3 Recurrent Neural LM

The written language can be viewed as a sequence of tokens. There are typically dependencies among these tokens. As an example, in the sentences 'The King was responsible himself', and 'Queen is in charge herself', the reflexive pronouns ('himself' and 'herself') grammatically depend on the gender of the subjects. The n-gram and feed-forward neural LMs both capture these dependencies by creating a context window around each token. However, window-based approaches fail to model long-range dependencies, as for the 'Queen-herself' dependency of the sentence 'The *Queen*, who criticized the King publicly on social media, was in fact responsible *herself* for the decision'.

The Recurrent Neural Network (RNN) makes it possible to model these long-range dependencies. The RNN adds a connection that references the previous hidden state h_{t-1} when calculating the hidden state of the current time step t, recursively defined as $\text{RNN}(x, h_{t-1}) \doteq tanh(W \times x + Q \times h_{t-1} + b_1)$. The matrices W and Q are the corresponding weights for the input x and the previous hidden state h_{t-1}. The RNN-based language model for the input

sentence $X = (x_1, x_2, x_3, ..., x_l)$ can be written in equations as:

$$\begin{aligned} h_t &= \text{RNN}(em(x_{t-1}), h_{t-1}) \\ s &= U \times h_t + b_2 \\ p(x_t|x_1, x_2, ..., x_{t-1}) &= \frac{e^{s_{x_t}}}{\sum_{\text{word}} e^{s_{\text{word}}}} \end{aligned} \tag{2.3}$$

The ability to pass information across an arbitrary number of consecutive time steps is the key strength of RNN-based LMs, which allows them to handle long-distance dependencies, and outperform n-gram and feed-forward LMs (Mikolov et al., 2010).

There is still a major issue in training the vanilla RNN model described above. It has been demonstrated that the gradient of the RNN's loss computed at each step, which back-propagates through all previous time steps, can diminish to zero, or explode exponentially (Pascanu et al., 2012). To prevent the exploding effect, the gradients are clipped according to a maximum norm. To resolve the vanishing gradient issue, and make a recurrent network remember longer steps, Chung et al. (2014) introduce Gated Recurrent Units (GRU) and compare it with Long Short-Term Memory (LSTM) networks introduced by Hochreiter and Schmidhuber (1997). Recently, the LSTM network has been widely used in various NLP tasks, which employs several internal gates to augment/clear its encoded information from the input sequence. The following equations summarize the computation of $\text{LSTM}(x, h_{t-1})$:

$$\begin{aligned} i_t &= sigmoid(W_i \times x + Q_i \times h_{t-1} + b_i) \\ f_t &= sigmoid(W_f \times x + Q_f \times h_{t-1} + b_f) \\ o_t &= sigmoid(W_o \times x + Q_o \times h_{t-1} + b_o) \\ n_t &= tanh(W_n \times x + Q_n \times h_{t-1} + b_n) \\ h_t &= o_t \cdot tanh(c_t) \quad c_t = f_t \cdot c_{t-1} + i_t \cdot n_t \end{aligned} \tag{2.4}$$

The i_t is the input gate rejecting or accepting the new memory n_t, and the forget vector f_t can clear the buffered memory c_{t-1}. The output vector o_t controls the emission of c_t as the output state h_t. The LSTM network is a dynamical system which creates shortcut paths for the gradients to be back-propagated to the earlier steps, so it allows the network to capture long-range dependencies. Despite the vanilla RNN which only multiplies the hidden

state to a matrix, the LSTM adds the new information to its buffer. This addition effectively creates these shortcut paths that bypass multiple temporal steps, and allow the error to be back-propagated without too quickly vanishing. Still, the LSTM can have exploding gradients (further details are provided by Hochreiter and Schmidhuber (1997) and Chung et al. (2014)).

Chapter 3

Conditional Sequence Modelling

In this chapter, we describe our techniques for mapping the input sequence $X = (x_1, x_2, x_3, ..., x_l)$ into the output sequence $Y = (y_1, y_2, y_3, ..., y_{l'})$ where each y_t is an output token. We first introduce the state-of-the-art models for transforming (encoding) the input X into a sequence of hidden vectors $H = (h_1, h_2, ..., h_l)$. We then investigate the following three discriminative decoding techniques for predicting the output sequence Y given H: Independent Prediction, Linear-Chain Conditional Random Fields, and decoder RNNs. We will study each of these methods in detail at both training and inference phases. The distribution $P(Y|X)$ formed by these methods is referred to as a conditional language model.

3.1 Encoders

Following Huang et al. (2015), the encoder of our sequence labeler employs a bi-directional RNN over the tokens in the sequence X. 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 (Figure 3.1). The hidden-dense layer applies a linear affine transformation to cut half the dimensions of its input, and then passes the result through a *tanh* activation function.

For sequence labeling, it has been shown that the prefix-suffix information of the words can be extracted using another character-level model, whether using RNNs (Lample et al., 2016) or Convolutional Neural Networks (Ma and

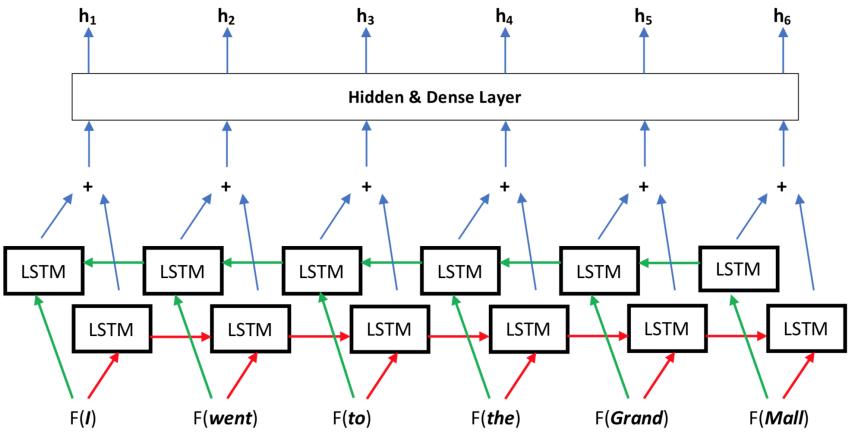

Figure 3.1: Time-unfolded encoder for labeling the sentence *'I went to the Grand Mall'*. F retrieves the computed feature vectors. The symbol '+' denotes the concatenation of the forward (left-to-right) and backward (right-to-left) outputs. The LSTM block receives the input & previous state to generate a new state.

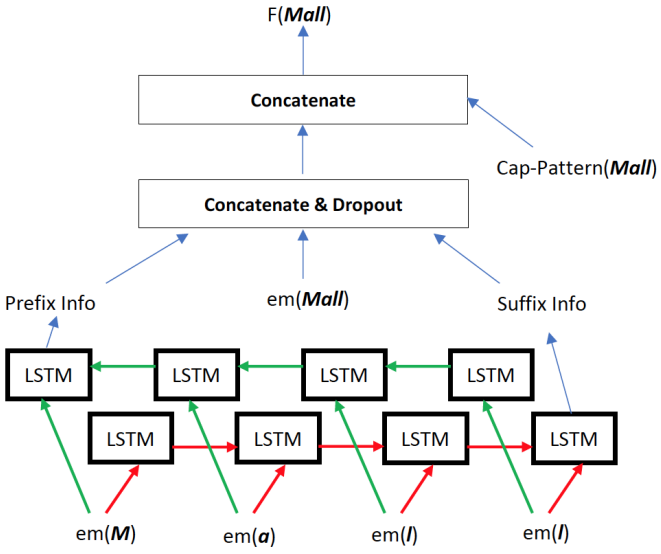

Figure 3.2: Sub-encoder for building the feature vector of 'Mall' used in the encoder of Figure 3.1. *em* retrieves word or character embeddings.

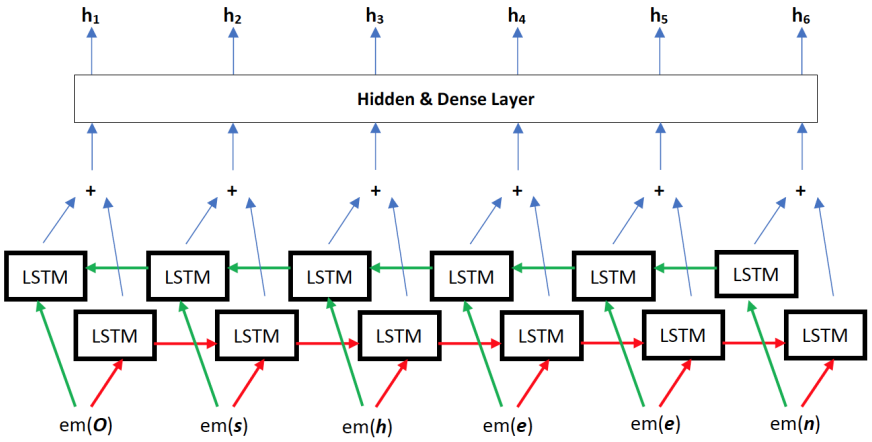

Figure 3.3: Time-unfolded encoder for transliterating *'Osheen'*. *em* retrieves character vector representations.

Hovy, 2016). This extra information is helpful for out-of-vocabulary words, and resemble word-shape patterns previously used in feature-based methods (Finkel et al., 2005). In the NER task, the model should learn this knowledge that several person names end with the suffix 'ie' (e.g. Ellie, Natalie, Sophie, Charlie, etc.) Therefore, we follow Lample et al. (2016), and build our context-independent word representation by combining a word-embedding matrix[1] with the outputs of another bi-directional RNN applied to each word's characters (Figure 3.2). The final states of the forward and backward character-level RNNs are concatenated to the word's embedding, and then passed through a dropout layer[2]. We then concatenate capitalization pattern indicators to these feature vectors (e.g. first letter is capital or all letters are capital).

Following Jadidinejad (2016), and Rosca and Breuel (2016), for transliteration, where we operate exclusively on the character level, we only apply a bi-directional RNN on the character representations, which are provided by a randomly initialized character-embedding matrix (Figure 3.3).

[1]The word embeddings are initialized using embeddings pre-trained on a large corpus.
[2]Masking out some dimensions during training for regularization (Srivastava et al., 2014).

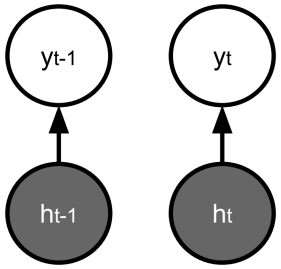

Figure 3.4: The Independent Prediction.

3.2 Independent Prediction (INDP)

Given an input X, we look for an output sequence $Y = (y_1, y_2, ..., y_l)$ where each y_t is an output token. In the encoder, we transform the input X into a sequence of hidden vectors $H = (h_1, h_2, ..., h_l)$ where each h_t observes all tokens in X. Given these vectors, the INDP method does not account for output dependencies at all. Instead, it independently predicts the output at time t by mapping h_t into the probability distribution $p_{\text{INDP}}(y_t|h_t)$ using a softmax layer:

$$\begin{aligned} s &= U \times h_t + b \\ p_{\text{INDP}}(y_t|h_t) &= \frac{e^{s_{y_t}}}{\sum_{label} e^{s_{label}}} \end{aligned} \tag{3.1}$$

This results in a sequence-level probability of $P(Y|X) = \prod_t p_{\text{INDP}}(y_t|h_t)$. The Figure 3.4 illustrates the probabilistic graphical representation of the INDP. During training, we maximize $p_{\text{INDP}}(y_t|h_t)$ for the gold label y_t. For the inference phase, we greedily select the output token with the highest probability at each step: $\hat{y}_t = \text{argmax}_{label}\, p_{\text{INDP}}(label|h_t)$. The INDP method is not applicable to the transduction task where the input and output sequences have different lengths.

3.3 Conditional Random Field (CRF)

In sequence labeling, there are typically dependencies between the output tokens. The most prominent and widely used approach to track these dependencies is the linear-chain CRF (Lafferty et al., 2001), which uses dynamic

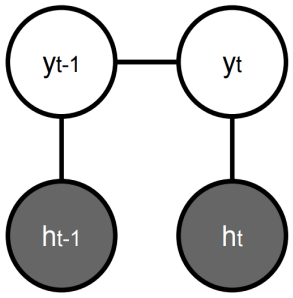

Figure 3.5: The Linear-Chain CRF.

Algorithm 1 Forward Algorithm

- Given: v: total number of labels, l: length of sequence
- Input: $E(\text{label}_i, \text{label}_j, t)$, and transition matrix T
- Initialize forward matrix $F[l, v]$
- For i=1 to v:
  - $F[1, i] = exp\{E(\text{label}_0, \text{label}_i, 1)\}$
- For t=2 to l:
  - For i=1 to v:
    - $F[t, i] = \sum_{j=1}^{v} F[t-1, j] \times exp\{E(\text{label}_j, \text{label}_i, t)\}$
- For i=1 to v:
  - $F[l, i] = F[l, i] \times exp\{T_{\text{label}_i, \text{label}_{l+1}}\}$
- $Z = \sum_{i=1}^{v} F[l, i]$

programming over an undirected graphical model (Figure 3.5) to maintain a well-defined probability distribution over the output sequences, effectively modelling $P(Y|X) = P_{\text{CRF}}(Y|H)$, where P_{CRF} hides first-order Markov dependencies over Y. The CRF can be used as a node over the LSTM-based encoder, while still allowing the system to train end-to-end (Huang et al., 2015).

Let $\psi(\text{label}_1, ..., \text{label}_l)$ be the score (energy) for an output sequence of length l, and $E(\text{label}_{t-1}, \text{label}_t, t)$ the scores of independent prediction of label_t,

and its transition from label_{t-1}, we then have:

$$\psi(\text{label}_1^l) = \left[\sum_{t=1}^{l} E(\text{label}_{t-1}, \text{label}_t, t)\right] + T_{\text{label}_l, \text{label}_{l+1}} \tag{3.2}$$
$$E(\text{label}_i, \text{label}_j, t) = T_{\text{label}_i, \text{label}_j} + S_{t, \text{label}_j}$$

The label_0 and label_{l+1} are the special start and end symbols. The square matrix T of size $v+2$ stores the transition score for two adjacent labels, where v denotes the total number of labels. The matrix S of size $l \times v$ keeps the emission scores at each step, where the row $S_t = U \times h_t + b$. We define the distribution $P_{\text{CRF}}(Y|H) = \frac{exp\{\psi(y_1,\ldots,y_l)\}}{Z}$, where $Z = \sum_{\text{label}_1^l} exp\{\psi(\text{label}_1^l)\}$ is our normalization constant enumerating over all possible output sequences.

The partition function Z can be efficiently computed in a linear-chain CRF using the Forward algorithm shown in Algorithm 1, where the array cell $F[t, i]$ stores the sum of exponentiated scores for different paths that predict the label i at time step t. Then, we have $Z = \sum_{i=1}^{v} F[l, i]$. During training, the loss -log$P_{\text{CRF}}(Y|H)$ is back-propagated to the matrix T, and to the encoder through matrix S. For inference, CRF employs the Viterbi algorithm shown in Algorithm 2 to predict the best output sequence $\hat{Y}$ that maximizes P_{CRF}.

As CRF operates over a neural encoder, it is necessary to train the network end-to-end processing the mini-batches of input instances in parallel. It is notable that the Forward algorithm for a single input sequence has a time complexity of $\theta(l \times v^2)$, and a memory complexity of $\theta(v^2)$ (because of matrix T, and $l \ll v$). With the recent computing resources (GPU cards), we can assume that the linear operations over matrices can be computed in a constant time. As a result, it is possible to compute the algorithm's recursion step with a single loop over the input sequence in the time complexity of $\theta(l)$, and memory complexity of $\theta(v^2)$. With the batch-processing of input sequences, the time complexity can remain $\theta(l)$, however the linear operations over matrices will require the memory complexity of $\theta(\text{batch size} \times v^2)$, due to the matrix broadcasting required for the summation of the matrices T and S.

In addition, the linear-chain CRF is not applicable to transduction tasks. In our experiments, in order to allow CRF to generate output of a different length from its input, we pad both sequences with extra end symbols up to a

Algorithm 2 Viterbi Algorithm

- Given: v: total number of labels, l: length of sequence
- Input: $E(\text{label}_i, \text{label}_j, t)$, and transition matrix T
- Initialize score matrix $V[l, v]$
- Initialize path back-pointer array $BP[l, v]$
- For i=1 to v:
  - $V[1, i] = exp\{E(\text{label}_0, \text{label}_i, 1)\}$
  - $BP[1, i] = \text{label}_0$
- For t=2 to l:
  - For i=1 to v:
    - $V[t, i] = \max_{j=1}^{v} V[t-1, j] \times exp\{E(\text{label}_j, \text{label}_i, t)\}$
    - $max_j = \text{argmax}_{j=1}^{v} V[t-1, j] \times exp\{E(\text{label}_j, \text{label}_i, t)\}$
    - $BP[t, i] = \text{label}_{max_j}$
- For i=1 to v:
  - $V[l, i] = V[l, i] \times exp\{T_{\text{label}_i, \text{label}_{l+1}}\}$
- $max_i = \text{argmax}_{i=1}^{v} V[l, i]$
- $bestLastLabel = \text{label}_{max_i}$
- Backtrace best labels by following BP starting from $bestLastLabel$

fixed maximum length, and let CRF decode until the end of the padded source sequence. It controls its target length by outputting padding tokens.

3.4 Decoder RNN

An alternative technique for predicting the output sequence Y is to use a decoder RNN on top of the encoder, as is typically done in Neural Machine Translation (NMT) (Sutskever et al., 2014), and has been done for CCG supertagging (Vaswani et al., 2016). The decoder RNN generates output tokens incrementally from left-to-right where it forms a distribution for each step based on the generated tokens of the previous steps, and the relevant source-side tokens. 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 relevant tokens of the input X for time t. For sequence labeling,

the source-to-target alignment is trivial, and c_t is provided directly by the encoder: $c_t = h_t$. For transliteration, where the input and output sequence lengths do not match, similar to the standard practice in NMT, an attention mechanism (Bahdanau et al., 2015) learns an alignment model that provides a scalar probability $\alpha_{t,t'}$ for each target position t and source position t', giving us the context vector $c_t = \sum_{t'} \alpha_{t,t'} h_{t'}$. The d_t is then recursively defined as: $d_t = \text{RNN}(d_{t-1}, c_{t-1}, em(y_{t-1}))$, where the previous d_{t-1} and c_{t-1} are both inputs to the decoder RNN, and em retrieves the feature vector of the previous output token y_{t-1}. Finally, a softmax layer is applied to the vectors d_t and c_t which defines the probability distribution $p_{\text{RNN}}(y_t|c_t, d_t)$, resulting in a sequence model of: $P(Y|X) = \prod_t p_{\text{RNN}}(y_t|X, y_1^{t-1}) = \prod_t p_{\text{RNN}}(y_t|c_t, d_t)$. The full computation of $p_{\text{RNN}}(y_t|X, y_1^{t-1})$ can be summarized in the following equations:

$$
\begin{aligned}
d_t &= \text{RNN}(d_{t-1}, c_{t-1}, em(y_{t-1})) \\
score(d_t, h_{t'}) &= d_t^T \times W' \times h_{t'} \qquad 1 \le t' \le l \\
\alpha_{t,t'} &= \frac{e^{score(d_t, h_{t'})}}{\sum_j e^{score(d_t, h_j)}} \qquad 1 \le t' \le l \\
c_t &= \sum_{t'=1}^{l} \alpha_{t,t'} h_{t'} \\
s &= U_2 \times tanh(U_1 \times \text{concat}\{c_t, d_t\} + b1) + b2 \\
p_{\text{RNN}}(y_t|X, y_1^{t-1}) &= \frac{e^{s_{y_t}}}{\sum_{\tilde{y}} e^{s_{\tilde{y}}}}
\end{aligned}
\tag{3.3}
$$

To learn the alignment model for transliteration, we use the soft-general attention mechanism of Luong et al. (2015). The decoder RNN requires the new parameters M': output embedding matrix retrieved by em, W': attention matrix mapping source hidden space to decoder hidden space, U_1, b_1, U_2, and $b2$: weights and biases of the softmax layer, and RNN(): the internal weights and biases of the RNN unit.

Similar to the INDP and CRF models, the encoder-decoder RNNs are trained with the maximum-likelihood objective $J_{ml}(\theta) = \sum_{X,Y} logP_\theta(Y|X)$. For training the network, the gold-standard previous token y_{t-1} is always fed into the decoder at time t (Figure 3.6). This procedure is known as Teacher Forcing (Goodfellow et al., 2016). At test time, we use the model's generated

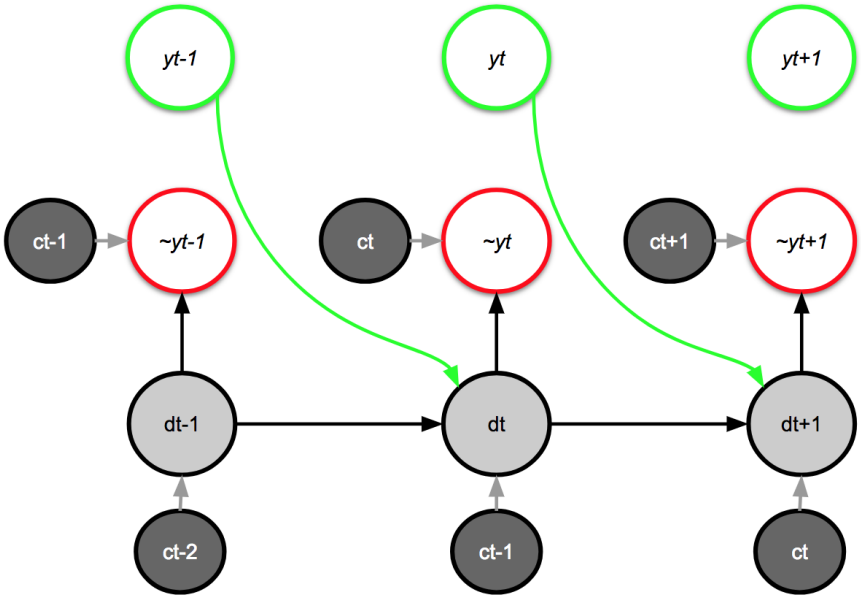

Figure 3.6: Teacher Forcing decoder RNN during maximum-likelihood training with the gold previous token y_{t-1}. We omit the dependency of the context vector c_t on the decoder hidden state d_t.

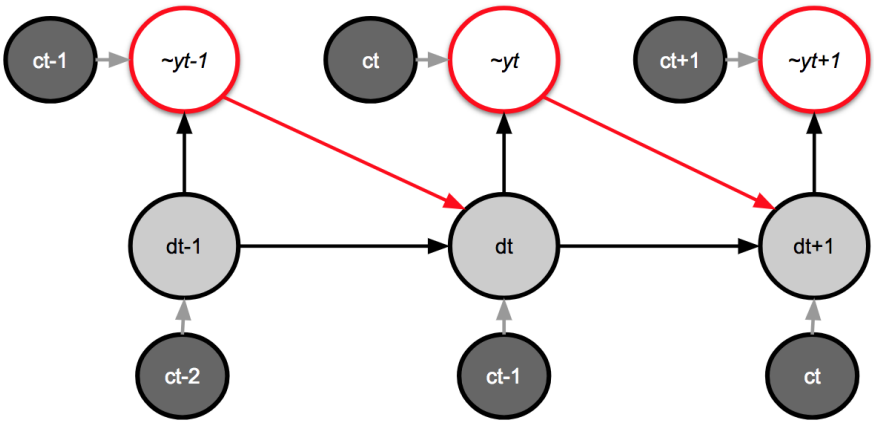

Figure 3.7: Decoder RNN at inference phase with the perviously generated token $_{\sim}y_{t-1}$.

output $\sim y_{t-1}$ (Figure 3.7). Because of this train-test mismatch, the network is not exposed to its own errors during training, and suffers from exposure bias (Bengio et al., 2015).

During the testing phase in the decoder RNN, we can use the greedy or beam search to generate the output tokens. In the greedy search, at each time step, we generate the token with the highest probability according the formed softmax distribution. In the beam search, for each time step, we keep a list of top b most probable sequences (b is the beam size). In the next time step, we generate b tokens for each of the sequences already in the beam list feeding their last generated token into the decoder RNN.[3] Then, out of $b \times b$ possible sequences, we select new top b sequences (based on the likelihood of each sequence), and update the beam list.

[3]We also need to feed the corresponding previous decoder state and context vector into the decoder RNN for each sequence in the beam list.

Chapter 4

Solutions for Exposure Bias

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. We set out to establish the decoder RNN as an attractive alternative to the CRF. In this chapter, we first discuss all prior works applied to resolve exposure bias, and then introduce our solution output-adjusted actor-critic training which adopts the actor-critic reinforcement learning, and specializes it for sequence labeling.

4.1 Prior Solutions

We review the following major techniques applied to resolve exposure bias in decoder RNNs: Scheduled Sampling, Beam-search Training, Professor Forcing, Reinforcement Learning, and Learning to Search. To our knowledge, none of these prior works has compared its method against CRF.

4.1.1 Scheduled Sampling (SS)

Bengio et al. (2015) introduce the notion of scheduled sampling, where the decoder RNN is gradually exposed to its own errors. In this approach, at each time step t with probability ϵ, we feed the ground-truth 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 consider this approach as one of our baselines. Although Scheduled Sampling has been shown to outperform the Teacher Forcing maximum-likelihood objective on several sequence-to-sequence tasks (Bengio et al., 2015), it has been demonstrated that as ϵ approaches zero, the model learns a biased conditional distribution which is different than the true distribution (Huszár, 2015).

4.1.2 Beam-search Training

Wiseman and Rush (2016) employ beam search in the training phase where the gold sequence remains at the top of the beam list at each step. In this approach, if the gold sequence is not ranked first, the method recognizes a violation, and its corresponding cost is back-propagated to the model's parameters. This method linearly increases the training time with respect to the beam size.

4.1.3 Professor Forcing

Lamb et al. (2016) introduce the Professor Forcing method based on the Generative Adversarial Networks (Goodfellow et al., 2014) to encourage the dynamics (parameters hidden space) of the decoder RNN to be the same during training and testing (the discriminator is fooled accordingly), however, they report lower results compared to the Teacher Forcing technique on shorter sequences with length less than 100.

4.1.4 Reinforcement Learning (RL)

Reinforcement learning has been applied to structured prediction tasks (Maes et al., 2009). One of the contributions in this dissertation is to take Maes's formalism of Structured Prediction Markov-Decision Process (SP-MDP), and apply it to deep neural architectures, whereas Maes et al. (2009) only investigated non-neural feature-based methods. Recently Ranzato et al. (2016) apply the REINFORCE algorithm (Williams, 1992) to Neural Machine Translation (NMT), to train the network with a reward derived from the BLEU score

of each generated sequence. Bahdanau et al. (2017) 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 (SC) training approach that does not require a critic model, which has been shown to outperform REINFORCE in the image captioning task. This method has also been applied to abstract summarization (Paulus et al., 2018).

The SC training is intended to represent the state-of-the-art in reinforcement learning for sequence-to-sequence models with sequence-level rewards, which will be another baseline in our experiments. The SC training samples a token based on the probability distribution formed over all tokens at each step, resulting in the sampled sequence $\tilde{Y} = (\tilde{y}_1, ..., \tilde{y}_{l'})$. The $\tilde{Y}$ is then compared to the greedily-sampled sequence $\hat{Y} = (\hat{y}_1, ..., \hat{y}_{l'})$. The SC defines a sequence-level cost (Δ) between the sampled sequences and the gold prediction Y, optimizing $J_{sc} = \sum_{X,Y} \Big(\Delta(\hat{Y}, Y) - \Delta(\tilde{Y}, Y)\Big) \times \log P_{\text{RNN}}(\tilde{Y}|X)$. If the sampled sequence $\tilde{Y}$ has a lower cost than the greedily-sampled sequence, the likelihood of $\tilde{Y}$ will be increased.

Unlike these previous works that apply reinforcement-learning techniques to optimize an available 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 tasks benefit from the binary rewards that are available at each step. In addition, Bahdanau et al. (2017), and Paulus et al. (2018) combine the Teacher Forcing maximum-likelihood objective with their proposed RL objectives, which require two forward computations in the decoder RNN, one for conditioning on the ground-truth labels, another for the RL objective without conditioning on the ground-truth labels. In this work, we will incorporate the supervision of the gold label into the actor-critic algorithm itself without any extra computation. Despite Bahdanau et al. (2017), our method employs a simpler critic architecture, where Bahdanau et al. (2017) use separate encoder-decoder RNNs for their critic model, which doubles the parameters of the network. Moreover, our approach will not rely on any schedules to pre-train the critic model. We will also present the first direct, controlled comparison between CRFs and any form of RNN.

4.1.5 Learning to Search

The Beam-search training is a special variant of the search-based methods designed for structured prediction problems (Daumé III, 2006; Daumé III et al., 2009; Ross et al., 2011; Chang et al., 2015). The Scheduled Sampling can also be considered as a special variant of this meta-learning approach. The main idea behind it is to transform the structured prediction into a simpler multi-class classification problem. To achieve this, Daumé III et al. (2009) propose in their SEARN algorithm to train a local classifier to predict each token sequentially, thus searching step by step in the big combinatorial space of the structured outputs. Recently Leblond et al. (2018) apply this SEARN algorithm to encoder-decoder RNNs (SEARNN) to resolve the exposure bias, where it generates several output sequences for a given input, feeding gold labels or the generated ones into the decoder RNN following reference or learned policies (or a mixture of both) at different time steps. The sequence-level costs are then defined for each of these searched outputs, and the local classifier (i.e. decoder RNN) is trained based on these costs. The SEARNN has been shown to perform slightly worse than the actor-critic method of Bahdanau et al. (2017) in NMT (Leblond et al., 2018).

4.2 Our Solution Output-Adjusted Actor-Critic Training

We frame the prediction of the output sequence as a sequential decision-making process, where the decoder RNN takes a series of actions without being conditioned on the ground-truth labels. We adopt the formalism of deterministic Markov Decision Process (MDP) designed for Structured Prediction by Maes et al. (2009). An SP-MDP environment is defined with four arguments $\mathcal{S}$, $\mathcal{A}$, T, and r where $\mathcal{S}$ is the finite set of all states, $\mathcal{A}$ is the set of all possible discrete actions, $T : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ is a transition function between states, and $r : \mathcal{S} \times \mathcal{A} \rightarrow R$ is the reward function. The following characteristics are also defined in SP-MDP which are specific for sequence labeling:

- **States**:

  Each state S_t is a pair of input sequence X and the partially-generated output sequence up to time step t: $S_t \doteq (X, [\hat{y}_1, ..., \hat{y}_{t-1}])$. The start and end states are denoted as $S_1 \doteq (X, [])$ and $S_{l'+1} \doteq (X, [\hat{y}_1, ..., \hat{y}_{l'}])$ where l' is the length of the complete output sequence. Therefore, for a given input X, we have a finite number of states.

- **Actions**:

  An action at each state is one possible output token (label). We have a finite set of possible labels (actions) for each step.

- **Transitions**:

  Transitions are deterministic. At state $S_t = (X, [\hat{y}_1, ..., \hat{y}_{t-1}])$, we take the action $A_t = \hat{y}_t$, and move to the state $S_{t+1} = (X, [\hat{y}_1, ..., \hat{y}_{t-1}, \hat{y}_t])$.

- **Rewards**:

  We employ per-position rewards for sequence labeling and transduction tasks. For the action $A_t = \hat{y}_t$, the per-position reward r_t is +1 if the action $\hat{y}_t$ is the same as the gold token y_t, and 0 otherwise.

In reinforcement learning, there is an actor (agent) that observes the current state of the environment, and takes an action accordingly. It then receives the corresponding reward for that action, and moves to the next state of the environment (Sutton and Barto, 1998). Such actor in the SP-MDP environment will have the following trajectory of states, actions, and rewards starting from the state S_1:

$$S_1,\ \hat{y}_1,\ r_1;\ S_2,\ \hat{y}_2,\ r_2;\ S_3,\ ...\ ;\ S_{l'},\ \hat{y}_{l'},\ r_{l'};\ S_{l'+1}$$

The goal of reinforcement learning is to make the actor choose actions that will maximize its future rewards. Accordingly, in the policy gradient methods, a stochastic policy (probability distribution) is defined over the actions. The notation $\pi_\theta(\hat{y}_t|S_t)$ denotes the policy for taking the action $\hat{y}_t$ conditioned on the current state of the environment S_t. The internal structure of the actor is given by the parameter set θ, which is used by the policy function π_θ. By

defining the return $G_t = \sum_{i=t}^{l'} r_i$ as the sum of future rewards after selecting the action $\hat{y}_t$, the learning objective $J_{pg}(\theta)$ is to update the parameter set θ such that the state-value function $V_{\pi_\theta}(S_1)$ for the start state is maximized, where:

$$J_{pg}(\theta) \doteq V_{\pi_\theta}(S_1) \tag{4.1}$$

$$V_{\pi_\theta}(S_t) \doteq E_{\pi_\theta}\left[G_t|S_t\right] = E_{\pi_\theta}\left[r_t + r_{t+1} + ... + r_{l'}|S_t\right] \tag{4.2}$$

The policy gradient theorem (Sutton et al., 1999) recommends the following objective for updating the parameter set θ (see Appendix A for a specific proof in the SP-MDP environment):

$$\frac{\partial J_{pg}(\theta)}{\partial \theta} = E_{\hat{y}_1,...,\hat{y}_{l'} \sim \pi_\theta}\left[\sum_{t=1}^{l'} \frac{\partial log(\pi_\theta(\hat{y}_t|S_t))}{\partial \theta}(G_t - b_{\theta'}(S_t))\right] \tag{4.3}$$

The actor needs to select a series of actions $\hat{y}_1, ..., \hat{y}_{l'}$ according to its current policy. The gradient of the objective $\frac{\partial J_{pg}(\theta)}{\partial \theta}$ suggests that we should update θ in a direction such that the log-likelihood of the action $\hat{y}_t$ is increased if it receives a higher return G_t. To reduce variance of the gradients, it is helpful to compare G_t against the baseline value $b_{\theta'}(S_t)$ which estimates the average of the returns previously received at state S_t. If the new action $\hat{y}_t$ has a return higher than the average, it should be preferred more by the policy function (i.e. its probability should be increased). Formally, $b_{\theta'}(S_t)$ can be a function estimating $V_{\pi_\theta}(S_t)$. We can estimate the expectation for the policy gradient objective with an average over multiple interactions of the actor within the SP-MDP: $\frac{\partial J_{pg}(\theta)}{\partial \theta} \approx \frac{1}{n} \sum_{(X_i, \hat{y}_{i_1}, ..., \hat{y}_{i_{l'}})} \sum_{t=1}^{l'_i} \frac{\partial log(\pi_\theta(\hat{y}_{i_t}|S_t))}{\partial \theta}(G_{i_t} - b_{\theta'}(S_{i_t}))$.

4.2.1 Actor-Critic for Decoder RNN in SP-MDP

We adopt the actor-critic (AC) version of the policy gradient objective (Sutton and Barto, 1998; Konda and Tsitsiklis, 2003; Mnih et al., 2016) to train the decoder RNN within the SP-MDP environment. 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 (return) for each prediction by comparing it to the gold-standard. The AC update modifies our RNN to improve credits at each step. This process also

Algorithm 3 Output-Adjusted 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:
  - $G_t = \sum_{i=0}^{n-1} [r_{t+i}] + V_{\theta'}(t+n)$
  - $\delta_t = G_t - V_{\theta'}(t)$
  - $a\delta_t = \text{adjust}(y_t, \hat{y}_t, \delta_t) \times \delta_t$
  - $loss_\theta = loss_\theta - a\delta_t \ln p_\theta(\hat{y}_t | X, \hat{y}_{t'<t})$
  - $loss_{\theta'} = loss_{\theta'} + \delta_t \times \delta_t$
- Back-propagate through $loss_\theta$ and $loss_{\theta'}$ as normal to update θ and θ' respectively
- Do not Back-propagate into G_t to update θ'

exposes the decoder to its own errors, alleviating exposure bias. Algorithm 3 provides pseudo code for the training process, which we expand upon in the following paragraphs.

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. We compute the sequence-level credit G_t for each decoding step using the multi-step Temporal Difference return (Sutton and Barto, 1998):

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

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

Input phrase	the	University	of	Alberta
Reference tags	O	Org	Org	Org
Model's prediction	O	Org	O	Loc
J_{ml}	$\log P(\text{O})$	$\log P(\text{Org}\|\text{O})$	$\log P(\text{Org}\|\text{O Org})$	$\log P(\text{Org}\|\text{O Org Org})$
J_{ac}	$1.9 \log P(\text{O})$	$0.9 \log P(\text{Org}\|\text{O})$	$-0.1 \log P(\text{O}\|\text{O Org})$	$-0.1 \log P(\text{Loc}\|\text{O Org O})$

Table 4.1: The formed probabilities by maximum-likelihood (J_{ml}) and actor-critic (J_{ac}) objectives for named entity tagging of the phrase 'the University of Alberta'. The advantage terms $\delta_t = 1.9, 0.9, \ldots$ given in J_{ac} are computed using $n = 4$, and $V(t) = 0.1$.

model that estimates the expected return $E[G_t]$, taking the context vector c_t and the decoder's hidden state d_t as input.[1] It is trained jointly alongside our decoder RNN, using a distinct optimizer (without back-propagating errors through c_t and d_t). With this critic in place, the update for the AC algorithm is defined as

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

where:

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

The AC update changes the prediction likelihood proportionally to the advantage δ_t of the token $\hat{y}_t$. Therefore, if $G_t > V_{\theta'}(t)$, the decoder should increase the likelihood. The AC error $\frac{\partial J_{ac}(\theta)}{\partial \theta}$ back-propagates only through the actor's prediction likelihood p_θ.

Table 4.1 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.

4.2.2 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 3 refers to this critic model. To

[1] We assume the current state of the SP-MDP environment is observed by the vectors c_t, which summarizes the relevant input tokens, and the decoder state d_t, which summarizes the previously generated output tokens.

learn the parameter set θ' in the critic, we use a semi-gradient update (Sutton and Barto, 1998). 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 $2\delta_t \frac{\partial(G_t - V_{\theta'}(t))}{\partial \theta'}$, we use the update $2\delta_t \frac{\partial V_{\theta'}(t)}{\partial \theta'}$. The Temporal Difference return G_t uses the critic's estimate $V_{\theta'}(t+n)$. The full gradient will cause a feedback loop as by doing so, we will allow G_t to match 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 G_t.

4.2.3 Output-Adjusted 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 adjusted advantage $a\delta_t$ as $\text{adjust}(y_t, \hat{y}_t, \delta_t) \times \delta_t$ where:

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

By setting the advantage $a\delta_t$ to 0, the adjust term effectively switches off the entire actor update when the advantage has the wrong polarity. Note that the critic is always updated. The output-adjusted training can also be interpreted as a combination of two advantages $\delta_t + \delta'_t$, where:

$$\delta'_t = \begin{cases} -\delta_t & \text{if} \quad \hat{y}_t = y_t \quad \& \quad \delta_t < 0 \\ -\delta_t & \text{if} \quad \hat{y}_t \neq y_t \quad \& \quad \delta_t > 0 \\ 0 & otherwise \end{cases}$$

The δ'_t adds the supervision of the gold label into the objective as it is always positive for the correctly selected tag, and negative for the wrongly selected token.

Similar to prior RL works (Ranzato et al., 2016; Bahdanau et al., 2017; Paulus et al., 2018), we pre-train the network using the Teacher Forcing maximum-likelihood objective (J_{ml}). We then continue training from the best model using our output-adjusted actor-critic objective. Figure 4.1 shows development set performance, starting from the same J_{ml}-pre-trained point, for our output-adjusted objective, as compared to standard actor-critic, REINFORCE with baseline, and normal REINFORCE on German NER. We observe that the output-adjusted training helps the model reach a higher point compared to all other objectives. As discussed in section 4.1.4, despite prior RL methods, the output-adjusted actor-critic objective does not require any schedule for pre-training the critic. It also avoids the necessity of combining the actor-critic objective with the Teacher Forcing maximum-likelihood training, as compared in Figure 4.1. After the pre-training phase, the related works still combine the maximum-likelihood training with their proposed RL objectives.

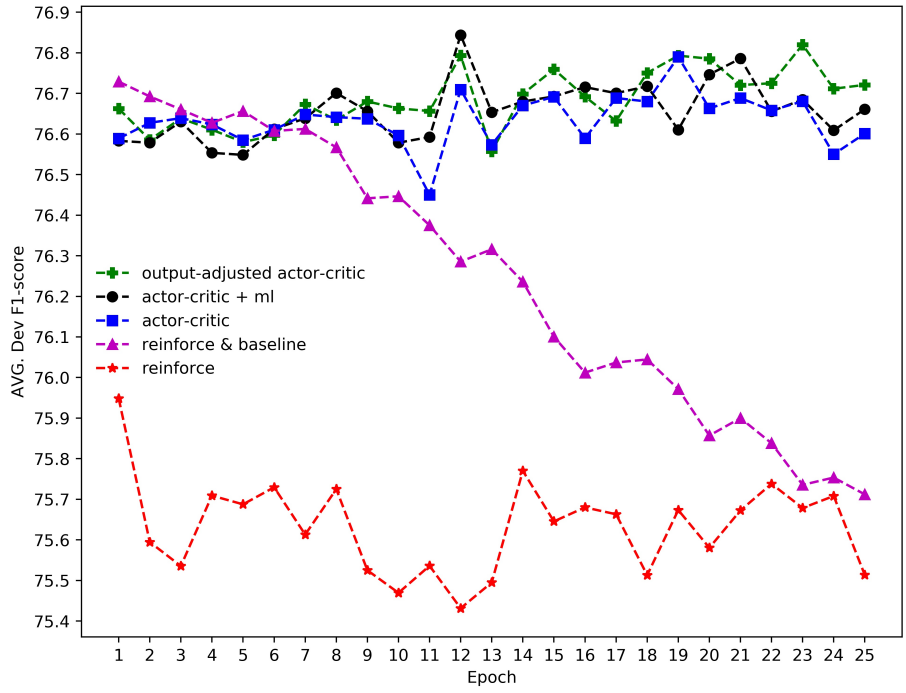

Figure 4.1: The output-adjusted actor-critic objective compared to alternative policy-gradient objectives on German NER with 17 possible output tags. All objectives are maximized using Gradient Ascent with a fixed step size of 0.5 for actor-critic, and 0.01 for REINFORCE objectives. For REINFORCE objectives, we set $n = l'$ in the Temporal Difference credits (i.e. sum all rewards until the end of sequence). All methods are trained 20 times with different random seeds using the same hyper-parameters.

Chapter 5

Experiments

In this chapter, we report our experiments comparing the following three main models:

- RNN: encoder-decoder RNN trained with the maximum-likelihood objective
- CRF: encoder RNN with CRF layer trained with maximum-likelihood objective
- AC-RNN: encoder-decoder RNN pre-trained with the maximum-likelihood objective, and fine-tuned with the output-adjusted actor-critic objective.

We comprehensively compare AC-RNN, CRF, and RNN on three tasks: NER tagging, CCG supertagging, and transliteration. We then compare the output-adjusted actor-critic objective with Scheduled Sampling on NER. Finally, we compare our training strategy to the Self-Critical method of Rennie et al. (2017).

5.1 Setup

5.1.1 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). Both datasets are annotated with 4 different entity types: 'Location', 'Organization', 'Person', and 'Miscellaneous' (e.g. events, nationalities, etc.). The Tables

Split	Sentences	LOC	ORG	PER	MISC
Training set	14987	7140	6321	6600	3438
Development set	3466	1837	1341	1842	922
Test set	3684	1668	1661	1617	702

Table 5.1: The number of sentences and annotated named entities on the CoNLL-2003 English dataset.

Split	Sentences	LOC	ORG	PER	MISC
Training set	12705	4363	2427	2773	2288
Development set	3068	1181	1241	1401	1010
Test set	3160	1035	773	1195	670

Table 5.2: The number of sentences and annotated named entities on the CoNLL-2003 German dataset.

5.1 and 5.2 provide the number of sentences and annotated entities for each dataset. As we have multi-word named entities (e.g. 'University of Alberta'), we employ the 'BILOU' tagging scheme[1] (Ratinov and Roth, 2009) to detect boundary of an entity in which 'B' indicates the beginning of an entity, 'I' is for interior words, 'L' marks the end of a span and 'U' represents single-word named entities. Overall, we have 17 output labels for each input token.

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. The train, dev, and test sets contain 38798, 1888, and 2327 sentences, respectively. We consider all the 1284 supertags appeared in the train set.

We use pre-trained, 100-dimensional *Glove* embeddings (section 2.2) for all English word-level tasks, 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

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

(EnTh) datasets of the NEWS-2018 shared task.[2] 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.

5.1.2 Implementation Details

We implement all our models in a shared code base using the PyTorch[3] framework (Paszke et al., 2017). The framework builds computational graphs dynamically during training, and has an auto-differentiation feature to back-propagate errors to all trainable parameters. The framework also provides various matrix mathematical operations, and includes the implementation of RNNs and different optimizers. Our developed tool with its implemented models is publicly available at `https://github.com/SaeedNajafi/ac-tagger`.

5.1.3 Training Details

For the experiments, our different models share the same encoder, using the same number of hidden units. Table 5.3 summarizes the hyper-parameters used in our experiments. The maximum-likelihood training is done with the Adam optimizer (Kingma and Ba, 2015) with a learning rate of 0.0005. The RL training is done with the mini-batch gradient ascent ($\theta = \theta + \alpha \frac{\partial J_{ac}(\theta)}{\partial \theta}$) using a fixed step size of 0.5 for NER & CCG, and 0.1 for transliteration experiments[4]. 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 model.

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 $\pm$ the standard deviations. Due to time constraints, for the translit-

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

[3]https://pytorch.org/

[4]We also tried Adam and RMSProp optimizers for the RL training, but both completely diverged.

Hyper-parameter	En-NER/De-NER	CCG	TL
char_embedding_size	32	32	128
output_embedding_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 5.3: The hyper-parameters used in the experiments.

eration experiments, we train each model only once.

5.1.4 Evaluation

We compute the standard evaluation metric for each task: entity-level F1-score[5] for NER, tagging accuracy for CCG, and word-level accuracy for transliteration. For the models with decoder RNNs, we report the results achieved using a beam search 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

5.2.1 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 3.2) as another baseline.

The results of the NER experiments are shown in Tables 5.4 and 5.5. We observe that by modelling 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. On German NER, we cannot

[5]A multi-word entity is predicted correctly if all of its words have the correct labels.

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 5.4: 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 5.5: Average entity-level F1-score for German NER on the CoNLL-2003 dataset.

replicate the CRF results of Lample et al. (2016), although we obtained their German word embeddings. We attribute this discrepancy to different preprocessing of the dataset. Moreover, AC-RNN significantly outperforms both RNN and CRF on both English and German test sets with the corresponding P values of 0.001 and 0.004 for RNN, and 0.003 and 0.016 for CRF. 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.

On CCG supertagging (Table 5.6), AC-RNN is significantly better than all other models with the P values of 0.019, 0.025, and 0.002, respectively, 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 5.7 shows the time and memory requirements for each method. We observe that, due to the large output vocabulary size of the task (1284

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 5.6: 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 5.7: The required GPU memory and execution time for one training epoch of the models on CCG supertagging using mini-batches of size 10.

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. The Forward algorithm of CRF runs out of memory with the mini-batch size of 16 on a 12-GB Graphical Processing Unit.

The transliteration results in Table 5.8 show that AC-RNN outperforms CRF (likely due to CRF's inability for predicting an output of a different length from its input), as well as RNN (likely due to its exposure bias). The transliteration experiments support our hypothesis that AC-RNN is more generally-applicable than CRF, and the improvements from the biased actor-critic training transfer to other tasks.

To confirm that our RNN baseline performs reasonably well, we validate our transliteration model against a standard NMT implementation as provided

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

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 5.9: The output-adjusted actor-critic training compared to Scheduled Sampling on German NER.

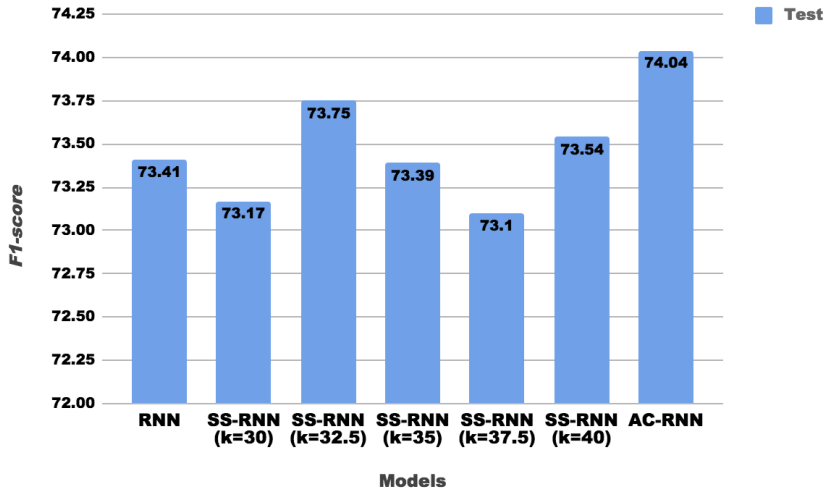

Figure 5.1: The sensitivity of Scheduled Sampling on the choice of sampling schedule on German NER. Higher k results in less sampling.

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 model with the attention mechanism.

5.2.2 Scheduled Sampling Comparisons

In the next experiment, we compare the output-adjusted actor-critic objective to Scheduled Sampling. We implement this approach using the inverse sigmoid schedule of Bengio et al. (2015) for annealing ϵ, and denote it as SS-RNN. The Table 5.9 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 with the P value of 0.040. This result supports our hypothesis that the reinforcement-learning solutions should outperform Scheduled Sampling, as the output-adjusted actor-critic training considers the entire sequence, whereas Scheduled Sampling addresses only exposure to the

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 5.10: The output-adjusted actor-critic objective compared to the Self-Critical policy training on the development sets of the NEWS-2018 shared task.

Model	Dev	Test
RNN	76.85 ±0.39	73.52 ±0.36
SC-RNN	76.71 ±0.27	73.50 ±0.42
AC-RNN	**77.10** ±0.29	**73.82** ±0.29

Table 5.11: The output-adjusted actor-critic training compared to the Self-Critical policy training on German NER.

immediately previous token. We also observe that Scheduled Sampling, unlike the output-adjusted actor-critic training, is highly sensitive to the choice of sampling schedule (Figure 5.1).

5.2.3 Self-Critical Comparisons

In our final experiment, we compare the output-adjusted actor-critic 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 transliteration results shown in Table 5.10 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. On German NER, the Self-Critical training cannot improve over RNN as shown in Table 5.11. These observations are aligned with our initial hypothesis that the reinforcement-learning techniques, applied to sequence labeling and transduction tasks, would benefit more from modelling the intermediate rewards, as is done with the Temporal Difference credits in the output-adjusted actor-critic training (section 4.2.1).

Chapter 6

Future Work

We have proposed an output-adjusted actor-critic algorithm to train encoder-decoder RNNs, which fine-tunes the model after the maximum-likelihood pre-training phase. A possible future direction is to propose a single objective which unifies these two separate phases. Although our output-adjusted method incorporates the supervision of the gold label into the actor-critic objective, it is still slower than the Teacher Forcing maximum-likelihood to train the network from the scratch. Recently, Norouzi et al. (2016) motivates a reward-augmented maximum-likelihood method which adds noise to the target labels according to their expected rewards. Still, this approach has neither been compared to the reinforcement learning objectives, nor studied against the CRF training.

In addition, other family of policy gradient methods can be investigated for resolving exposure bias. We can study the similarity between the natural policy gradient (Kakade, 2002), and our output-adjusted actor-critic applied over the maximum-likelihood objective. It is also worth investigating the off-policy actor-critic models (Degris et al., 2012) which are more general than the Self-Critical training. The SC method can be considered an off-policy approach where it utilizes a behavioural policy (i.e. sampling) to generate the sampled sequence, and the learning is guided with the greedily-sampled sequence (given by the greedy target policy).

Finally, the CRF models should be adapted for tasks with large output vocabularies. Recently, Mensch and Blondel (2018) smooth the max oper-

ations used in the Viterbi algorithm of CRF for NER task. Relaxing the dynamic-programming computations used in the CRF can reduce its memory complexity.

Chapter 7

Conclusion

We have proposed an output-adjusted actor-critic algorithm to train encoder-decoder RNNs for sequence labeling and transduction tasks. Though related reinforcement-learning algorithms have previously been applied to sequence-to-sequence tasks, 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

Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. 2016. Linear algebraic structure of word senses, with applications to polysemy. *CoRR*, abs/1601.03764.

Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron C. Courville, and Yoshua Bengio. 2017. An actor-critic algorithm for sequence prediction. ICLR.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. ICLR.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 1171–1179, Cambridge, MA, USA. MIT Press.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155.

Akash Bharadwaj, David Mortensen, Chris Dyer, and Jaime Carbonell. 2016. Phonologically aware neural model for named entity recognition in low resource transfer settings. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1462–1472. Association for Computational Linguistics.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daume, and John Langford. 2015. Learning to search better than your teacher. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2058–2066, Lille, France. PMLR.

Jason Chiu and Eric Nichols. 2016. Named entity recognition with bidirectional lstm-cnns. *Transactions of the Association of Computational Linguistics*, 4:357–370.

Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*.

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*.

Aron Culotta, David Kulp, and Andrew McCallum. 2005. Gene prediction with conditional random fields. *University of Massachusetts, Amherst, Tech. Rep. UM-CS-2005-028*.

Hal Daumé III. 2006. *Practical Structured Learning Techniques for Natural Language Processing*. Ph.D. thesis, University of Southern California, Los Angeles, CA.

Hal Daumé III, John Langford, and Daniel Marcu. 2009. Search-based structured prediction.

Thomas Degris, Martha White, and Richard S. Sutton. 2012. Off-policy actor-critic. ICML'12.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 363–370. Association for Computational Linguistics.

Dan Gillick, Cliff Brunk, Oriol Vinyals, and Amarnag Subramanya. 2016. Multilingual language processing from bytes. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1296–1306. Association for Computational Linguistics.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 2672–2680, Cambridge, MA, USA. MIT Press.

Roman Grundkiewicz and Kenneth Heafield. 2018. Neural machine translation techniques for named entity transliteration. In *Proceedings of the Seventh Named Entities Workshop*, pages 89–94. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, pages 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.

Ferenc Huszár. 2015. How (not) to train your generative model: Scheduled sampling, likelihood, adversary? *arXiv preprint arXiv:1511.05101*.

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

Daniel Jurafsky and James H. Martin. 2000. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 1st edition. Prentice Hall PTR, Upper Saddle River, NJ, USA.

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

Sham M Kakade. 2002. A natural policy gradient. In *Advances in neural information processing systems*, pages 1531–1538.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. ICLR.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. 2017. Opennmt: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*, pages 67–72. Association for Computational Linguistics.

Kevin Knight and Jonathan Graehl. 1998. Machine transliteration. *Computational Linguistics*, 24(4).

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

Onur Kuru, Ozan Arkan Can, and Deniz Yuret. 2016. Charner: Character-level named entity recognition. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 911–921. The COLING 2016 Organizing Committee.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Alex M Lamb, Anirudh Goyal ALIAS PARTH GOYAL, Ying Zhang, Saizheng Zhang, Aaron C Courville, and Yoshua Bengio. 2016. Professor forcing: A new algorithm for training recurrent networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4601–4609. Curran Associates, Inc.

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.

Rémi Leblond, Jean-Baptiste Alayrac, Anton Osokin, and Simon Lacoste-Julien. 2018. SEARNN: Training RNNs with global-local losses. In *International Conference on Learning Representations*.

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.

Jiwei Li, Will Monroe, Tianlin Shi, Alan Ritter, and Dan Jurafsky. 2017. Adversarial learning for neural dialogue generation. *CoRR*, abs/1701.06547.

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.

Arthur Mensch and Mathieu Blondel. 2018. Differentiable dynamic programming for structured prediction and attention. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3459–3468, StockholmsmÃ€ssan, Stockholm Sweden. PMLR.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 3111–3119, USA. Curran Associates Inc.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA. PMLR.

Seungwhan Moon, Leonardo Neves, and Vitor Carvalho. 2018. Multimodal named entity recognition for short social media posts. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 852–860. Association for Computational Linguistics.

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.

Saeed Najafi, Colin Cherry, and Grzegorz Kondrak. 2018a. Sequence labeling and transduction with output-adjusted actor-critic training of RNNs. In preparation.

Saeed Najafi, Bradley Hauer, Rashed Rubby Riyadh, Leyuan Yu, and Grzegorz Kondrak. 2018b. Combining neural and non-neural methods for low-resource morphological reinflection. In *Proceedings of the CoNLL–SIGMORPHON 2018 Shared Task: Universal Morphological Reinflection*, Brussels, Belgium. Association for Computational Linguistics.

Saeed Najafi, Bradley Hauer, Rashed Rubby Riyadh, Leyuan Yu, and Grzegorz Kondrak. 2018c. Comparison of assorted models for transliteration. In *Proceedings of the Seventh Named Entities Workshop*, pages 84–88. Association for Computational Linguistics.

Garrett Nicolai, Colin Cherry, and Grzegorz Kondrak. 2015. Morpho-syntactic regularities in continuous word representations: A multilingual study. In *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*, pages 129–134. Association for Computational Linguistics.

Garrett Nicolai, Saeed Najafi, and Grzegorz Kondrak. 2018. String transduction with target language models and insertion handling. In *Proceedings of the 15th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, Brussels, Belgium.

Mohammad Norouzi, Samy Bengio, zhifeng Chen, Navdeep Jaitly, Mike Schuster, Yonghui Wu, and Dale Schuurmans. 2016. Reward augmented maximum likelihood for neural structured prediction. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 1723–1731. Curran Associates, Inc.

Sebastian Nowozin and Christoph H. Lampert. 2011. Structured learning and prediction in computer vision. *Found. Trends. Comput. Graph. Vis.*, 6(3–4):185–365.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2012. Understanding the exploding gradient problem. *CoRR*.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. In *NIPS-W*.

Romain Paulus, Caiming Xiong, and Richard Socher. 2018. A deep reinforced model for abstractive summarization. ICLR.

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. 2016. Sequence level training with recurrent neural networks. ICLR.

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.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Emma Strubell, Patrick Verga, David Belanger, and Andrew McCallum. 2017. Fast and accurate entity recognition with iterated dilated convolutions. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2670–2680. Association for Computational Linguistics.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc.

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

Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 1057–1063, Cambridge, MA, USA. MIT Press.

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.

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3-4):229–256.

Sam Wiseman and Alexander M. Rush. 2016. Sequence-to-sequence learning as beam-search optimization. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1296–1306. Association for Computational Linguistics.

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

Mingbin Xu, Hui Jiang, and Sedtawut Watcharawittayakul. 2017. A local detection approach for named entity recognition and mention detection. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1237–1247. Association for Computational Linguistics.

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.

Zhilin Yang, Ruslan Salakhutdinov, and William W. Cohen. 2016. Multi-task cross-lingual sequence tagging from scratch. *CoRR*, abs/1603.06270.

Zheng Yuan and Ted Briscoe. 2016. Grammatical error correction using neural machine translation. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 380–386. Association for Computational Linguistics.

Appendix A

A.1 Policy Gradient Theorem in SP-MDP

Theorem A.1.1. *In the SP-MDP environment with deterministic transitions between states, for the input sequence X, the output sequence of length l', and the policy $\pi_\theta(\hat{y}_t|S_t)$, we have $V_{\pi_\theta}(S_1) = \sum_{t=1}^{l'} \sum_{\hat{y}_1,...,\hat{y}_t} \pi_\theta(\hat{y}_1, ..., \hat{y}_t|X) r_t$, where r_t is the reward for selecting the possible action $\hat{y}_t$.*

Proof. In the SP-MDP with deterministic transitions, for a given input X, states depend only on the previously selected actions. By the definition $V_{\pi_\theta}(S_1) \doteq E_{\pi_\theta}[G_1|S_1] = E_{\pi_\theta}[r_1 + r_2 + ... + r_{l'}|S_1]$, we have:

$$
\begin{aligned}
V_{\pi_\theta}(S_1) = E_{\pi_\theta}[r_1|S_1] + E_{\pi_\theta}[r_2|S_1] + ... + E_{\pi_\theta}[r_{l'}|S_1] = \\
\sum_{\hat{y}_1} \pi_\theta(\hat{y}_1|X) r_1 + \sum_{\hat{y}_1} \pi_\theta(\hat{y}_1|X) \sum_{\hat{y}_2} \pi_\theta(\hat{y}_2|\hat{y}_1, X) r_2 + ... \\
+ \sum_{\hat{y}_1} \pi_\theta(\hat{y}_1|X) \sum_{\hat{y}_2} \pi_\theta(\hat{y}_2|\hat{y}_1, X) \sum_{\hat{y}_3} ... \times \sum_{\hat{y}_{l'}} \pi_\theta(\hat{y}_{l'}|\hat{y}_1, ..., \hat{y}_{l'-1}, X) r_{l'} = \\
\sum_{\hat{y}_1} \pi_\theta(\hat{y}_1|X) r_1 + \sum_{\hat{y}_1} \sum_{\hat{y}_2} \pi_\theta(\hat{y}_2|\hat{y}_1, X) \pi_\theta(\hat{y}_1|X) r_2 + ... \\
+ \sum_{\hat{y}_1} \sum_{\hat{y}_2} ... \sum_{\hat{y}_{l'}} \pi_\theta(\hat{y}_{l'}|\hat{y}_1, ..., \hat{y}_{l'-1}, X) \times ... \times \pi_\theta(\hat{y}_2|\hat{y}_1, X) \pi_\theta(\hat{y}_1|X) r_{l'} = \\
\sum_{t=1}^{l'} \sum_{\hat{y}_1,...,\hat{y}_t} \pi_\theta(\hat{y}_t|\hat{y}_1, ..., \hat{y}_{t-1}, X) \times ... \times \pi_\theta(\hat{y}_2|\hat{y}_1, X) \pi_\theta(\hat{y}_1|X) r_t = \\
\sum_{t=1}^{l'} \sum_{\hat{y}_1,...,\hat{y}_t} \pi_\theta(\hat{y}_1, ..., \hat{y}_t|X) r_t
\end{aligned}
$$

□

Theorem A.1.2. *For the objective function $J_{pg}(\theta) \doteq V_{\pi_\theta}(S_1)$ in the SP-MDP*

environment with the input sequence X, the output sequence of length l', and the policy $\pi_\theta(\hat{y}_t|S_t)$, we have $\frac{\partial J_{pg}(\theta)}{\partial \theta} = E_{\hat{y}_1,...,\hat{y}_{l'}\sim\pi_\theta}[\sum_{t=1}^{l'} \frac{\partial log(\pi_\theta(\hat{y}_t|\hat{y}_1,...,\hat{y}_{t-1},X))}{\partial \theta}(G_t)]$.

Proof. We can use the result of the Theorem A.1.1 to calculate $\frac{\partial J_{pg}(\theta)}{\partial \theta} = \frac{\partial V_{\pi_\theta}(S_1)}{\partial \theta}$:

$$\frac{\partial J_{pg}(\theta)}{\partial \theta} = \frac{\partial V_{\pi_\theta}(S_1)}{\partial \theta} = \sum_{t=1}^{l'} \sum_{\hat{y}_1,...,\hat{y}_t} \frac{\partial \pi_\theta(\hat{y}_1,...,\hat{y}_t|X)}{\partial \theta} r_t =$$

$$\sum_{t=1}^{l'} \sum_{\hat{y}_1,...,\hat{y}_t} \pi_\theta(\hat{y}_1,...,\hat{y}_t|X) \times [\frac{\partial \pi_\theta(\hat{y}_1,...,\hat{y}_t|X)}{\partial \theta} \times \frac{1}{\pi_\theta(\hat{y}_1,...,\hat{y}_t|X)} \times r_t] =$$

$$\sum_{t=1}^{l'} \sum_{\hat{y}_1,...,\hat{y}_t} \pi_\theta(\hat{y}_1,...,\hat{y}_t|X) \times [\frac{\partial log(\pi_\theta(\hat{y}_1,...,\hat{y}_t|X))}{\partial \theta} r_t] \times 1 =$$

$$\sum_{t=1}^{l'} \sum_{\hat{y}_1,...,\hat{y}_t} \pi_\theta(\hat{y}_1,...,\hat{y}_t|X) \times [\frac{\partial log(\pi_\theta(\hat{y}_1,...,\hat{y}_t|X))}{\partial \theta} r_t] \times \sum_{\hat{y}_{t+1},...,\hat{y}_{l'}} \pi_\theta(\hat{y}_{t+1},...,\hat{y}_{l'}|\hat{y}_1,...,\hat{y}_t,X) =$$

$$\sum_{t=1}^{l'} \sum_{\hat{y}_1,...,\hat{y}_t} \sum_{\hat{y}_{t+1},...,\hat{y}_{l'}} \pi_\theta(\hat{y}_{t+1},...,\hat{y}_{l'}|\hat{y}_1,...,\hat{y}_t,X) \times \pi_\theta(\hat{y}_1,...,\hat{y}_t|X) \times [\frac{\partial log(\pi_\theta(\hat{y}_1,...,\hat{y}_t|X))}{\partial \theta} r_t] =$$

$$\sum_{t=1}^{l'} \sum_{\hat{y}_1,...,\hat{y}_{l'}} \pi_\theta(\hat{y}_1,...,\hat{y}_{l'}|X) \times [\frac{\partial log(\pi_\theta(\hat{y}_1,...,\hat{y}_t|X))}{\partial \theta} r_t] =$$

$$\sum_{t=1}^{l'} E_{\hat{y}_1,...,\hat{y}_{l'}\sim\pi_\theta}[\frac{\partial log(\pi_\theta(\hat{y}_1,...,\hat{y}_t|X))}{\partial \theta} r_t] =$$

$$E_{\hat{y}_1,...,\hat{y}_{l'}\sim\pi_\theta}[\frac{\partial log(\pi_\theta(\hat{y}_1|X))}{\partial \theta} r_1] + ... + E_{\hat{y}_1,...,\hat{y}_{l'}\sim\pi_\theta}[\frac{\partial log(\pi_\theta(\hat{y}_1,...,\hat{y}_{l'}|X))}{\partial \theta} r_{l'}] =$$

$$E_{\hat{y}_1,...,\hat{y}_{l'}\sim\pi_\theta}[\sum_{t=1}^{l'} \frac{\partial log(\pi_\theta(\hat{y}_1,...,\hat{y}_t|X))}{\partial \theta} r_t] =$$

$$E_{\hat{y}_1,...,\hat{y}_{l'}\sim\pi_\theta}[\sum_{t=1}^{l'} r_t \sum_{t'=1}^{t} \frac{\partial log(\pi_\theta(\hat{y}_{t'}|\hat{y}_1,...,\hat{y}_{t'-1},X))}{\partial \theta}] =$$

$$E_{\hat{y}_1,...,\hat{y}_{l'}\sim\pi_\theta}[\sum_{t=1}^{l'} \sum_{t'=1}^{t} \frac{\partial log(\pi_\theta(\hat{y}_{t'}|\hat{y}_1,...,\hat{y}_{t'-1},X))}{\partial \theta} r_t] =$$

$$E_{\hat{y}_1,...,\hat{y}_{l'}\sim\pi_\theta}[\sum_{t=1}^{l'} \frac{\partial log(\pi_\theta(\hat{y}_t|\hat{y}_1,...,\hat{y}_{t-1},X))}{\partial \theta} \sum_{t'=t}^{l'} r_{t'}] =$$

$$E_{\hat{y}_1,...,\hat{y}_{l'}\sim\pi_\theta}[\sum_{t=1}^{l'} \frac{\partial log(\pi_\theta(\hat{y}_t|\hat{y}_1,...,\hat{y}_{t-1},X))}{\partial \theta}(G_t)]$$

□

Theorem A.1.3. *In the SP-MDP with the input sequence X, the output sequence of length l', and the policy $\pi_\theta(\hat{y}_t|S_t)$, for any function $b_{\theta'}(S_t)$ which depends only on X, and the previously selected actions $\hat{y}_1, ..., \hat{y}_{t-1}$, we have $E_{\hat{y}_1,...,\hat{y}_{l'}\sim\pi_\theta}[\sum_{t=1}^{l'} \frac{\partial log(\pi_\theta(\hat{y}_t|\hat{y}_1,...,\hat{y}_{t-1},X))}{\partial\theta}(b_{\theta'}(S_t))] = 0$.*

Proof.

$$E_{\hat{y}_1,...,\hat{y}_{l'}\sim\pi_\theta}[\sum_{t=1}^{l'} \frac{\partial log(\pi_\theta(\hat{y}_t|\hat{y}_1,...,\hat{y}_{t-1},X))}{\partial\theta}(b_{\theta'}(\hat{y}_1,...,\hat{y}_{t-1},X))] =$$

$$\sum_{t=1}^{l'} E_{\hat{y}_1,...,\hat{y}_{l'}\sim\pi_\theta}[\frac{\partial log(\pi_\theta(\hat{y}_t|\hat{y}_1,...,\hat{y}_{t-1},X))}{\partial\theta}(b_{\theta'}(\hat{y}_1,...,\hat{y}_{t-1},X))] =$$

$$\sum_{t=1}^{l'} \sum_{\hat{y}_1,...,\hat{y}_{l'}} \pi_\theta(\hat{y}_1,...,\hat{y}_{l'}|X) \times [\frac{\partial log(\pi_\theta(\hat{y}_t|\hat{y}_1,...,\hat{y}_{t-1},X))}{\partial\theta}(b_{\theta'}(\hat{y}_1,...,\hat{y}_{t-1},X))] =$$

$$\sum_{t=1}^{l'} \sum_{\hat{y}_1,...,\hat{y}_{t-1}} b_{\theta'}(S_t)\pi_\theta(\hat{y}_1,...,\hat{y}_{t-1}|X) \times \sum_{\hat{y}_t,...,\hat{y}_{l'}} \pi_\theta(\hat{y}_t,...,\hat{y}_{l'}|S_t) \times \frac{\partial log(\pi_\theta(\hat{y}_t|S_t))}{\partial\theta} =$$

$$\sum_{t=1}^{l'} \sum_{\hat{y}_1,...,\hat{y}_{t-1}} b_{\theta'}(S_t)\pi_\theta(\hat{y}_1,...,\hat{y}_{t-1}|X) \times \sum_{\hat{y}_t} \pi_\theta(\hat{y}_t|S_t) \times \frac{\partial log(\pi_\theta(\hat{y}_t|S_t))}{\partial\theta} \times$$

$$\sum_{\hat{y}_{t+1},...,\hat{y}_{l'}} \pi_\theta(\hat{y}_{t+1},...,\hat{y}_{l'}|S_{t+1}) =$$

$$\sum_{t=1}^{l'} \sum_{\hat{y}_1,...,\hat{y}_{t-1}} b_{\theta'}(S_t)\pi_\theta(\hat{y}_1,...,\hat{y}_{t-1}|X) \times \sum_{\hat{y}_t} \frac{\partial \pi_\theta(\hat{y}_t|S_t)}{\partial\theta} \times 1 =$$

$$\sum_{t=1}^{l'} \sum_{\hat{y}_1,...,\hat{y}_{t-1}} b_{\theta'}(S_t)\pi_\theta(\hat{y}_1,...,\hat{y}_{t-1}|X) \times \frac{\partial E_{\hat{y}_t\sim\pi_\theta}[1]}{\partial\theta} =$$

$$\sum_{t=1}^{l'} \sum_{\hat{y}_1,...,\hat{y}_{t-1}} b_{\theta'}(S_t)\pi_\theta(\hat{y}_1,...,\hat{y}_{t-1}|X) \times \frac{\partial 1}{\partial\theta} =$$

$$\sum_{t=1}^{l'} \sum_{\hat{y}_1,...,\hat{y}_{t-1}} b_{\theta'}(S_t)\pi_\theta(\hat{y}_1,...,\hat{y}_{t-1}|X) \times 0 = 0$$

□