Structured Neural Models for Coreference and Generation

a dissertation presented
by
Samuel Joshua Wiseman
to
The John A. Paulson School of Engineering and Applied Sciences

in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in the subject of
Computer Science

Harvard University
Cambridge, Massachusetts
May 2018

Dissertation Advisor: Professor Stuart M. Shieber
Professor Alexander M. Rush

Samuel Joshua Wiseman

Structured Neural Models for Coreference and Generation

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Morbi commodo, ipsum sed pharetra gravida, orci magna rhoncus neque, id pulvinar odio lorem non turpis. Nullam sit amet enim. Suspendisse id velit vitae ligula volutpat condimentum. Aliquam erat volutpat. Sed quis velit. Nulla facilisi. Nulla libero. Vivamus pharetra posuere sapien. Nam consectetuer. Sed aliquam, nunc eget euismod ullamcorper, lectus nunc ullamcorper orci, fermentum bibendum enim nibh eget ipsum. Donec porttitor ligula eu dolor. Maecenas vitae nulla consequat libero cursus venenatis. Nam magna enim, accumsan eu, blandit sed, blandit a, eros.

Quisque facilisis erat a dui. Nam malesuada ornare dolor. Cras gravida, diam sit amet rhoncus ornare, erat elit consectetuer erat, id egestas pede nibh eget odio. Proin tincidunt, velit vel porta elementum, magna diam molestie sapien, non aliquet massa pede eu diam. Aliquam iaculis. Fusce et ipsum et nulla tristique facilisis. Donec eget sem sit amet ligula viverra gravida. Etiam vehicula urna vel turpis. Suspendisse sagittis ante a urna. Morbi a est quis orci consequat rutrum. Nullam egestas feugiat felis. Integer adipiscing semper ligula. Nunc molestie, nisl sit amet cursus convallis, sapien lectus pretium metus, vitae pretium enim wisi id lectus. Donec vestibulum. Etiam vel nibh. Nulla facilisi. Mauris pharetra. Donec augue. Fusce ultrices, neque id dignissim ultrices, tellus mauris dictum elit, vel lacinia enim metus eu nunc.

Contents

Listing of figures

This is the dedication.

Acknowledgments

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Morbi commodo, ipsum sed pharetra gravida, orci magna rhoncus neque, id pulvinar odio lorem non turpis. Nullam sit amet enim. Suspendisse id velit vitae ligula volutpat condimentum. Aliquam erat volutpat. Sed quis velit. Nulla facilisi. Nulla libero. Vivamus pharetra posuere sapien. Nam consectetuer. Sed aliquam, nunc eget euismod ullamcorper, lectus nunc ullamcorper orci, fermentum bibendum enim nibh eget ipsum. Donec porttitor ligula eu dolor. Maecenas vitae nulla consequat libero cursus venenatis. Nam magna enim, accumsan eu, blandit sed, blandit a, eros.

1

Introduction

Natural Language Processing has been experiencing a period of tremendous empirical success, with the state of the art on nearly all core NLP tasks significantly improved over the past several years. Much of this success is due to the widespread adoption of deep learning methods by NLP researchers. Deep learning has brought with it models and techniques, as well as philosophical views on how problems ought to be modeled. In the former category, word and feature embeddings as well as various forms of neural network have allowed NLP models to automatically learn powerful representations of linguistic data, which have turned out to be incredibly important in improving performance. In the latter category, deep learning practitioners have shown a preference for minimizing the amount of hand-engineering that goes into preparing the data for a model's consumption (in particular, in the form of hand-engineering features), as well as for neural networks that are "global" models of whatever phenomenon is being modeled. For instance, there has been a preference for learning global representations of sentences, documents, and parse trees, often by embedding the entire object, using a recurrent or convolutional neural network.

This preference for global models of complicated, linguistic objects has come at the expense of making *prediction* of these objects intractable; it becomes intractable to produce the highest scoring sentence, document, or parse tree when its score depends globally on the entire structure. Thus, the adoption of deep learning methods poses a challenge for *structured prediction*, the class of prediction problems which involves producing whole structures (such as sentences, documents, and parse trees), rather than single labels or real values. In particular, the use of global, neural models necessitates the use of structured prediction approaches that do only approximate inference; that is, that can only approximately find the highest scoring structures. While there are a variety of approaches to doing structured prediction when only approximate inference is possible, the main argument of the first part of this thesis is, in essence, that *search-based* structured prediction and recurrent neural networks are a match made in heaven.

Search-based structured prediction (reviewed in Chapter 2) involves training a model to incrementally search for an optimal structure, one part at a time. In order to do so effectively, a search-based structured prediction algorithm must learn to predict the next part to add to the partial structure being constructed, conditioned on all the parts already added. This history of partial predictions can be challenging to represent, especially with discrete features, since it can be quite long, and vary in size from example to example. Recurrent neural networks (RNNs), however, are an extremely natural choice to compress histories into a single vector; they require little feature engineering, naturally handle variable-length histories, and they require no independence assumptions about the components of the history, giving us a global model. Adopting this approach to neural structured prediction, we show we can obtain impressive performance on a variety of structured, natural language tasks, such as coreference resolution, word-ordering, dependency parsing, and machine translation.

While search-based structured prediction is natural for certain types of problems, it appears to be inadequate for one of the more pressing structured prediction problems being tackled with neural networks, namely, long-form text generation. In the final chapter of this thesis we examine this issue more closely, introducing a new dataset for database-to-document generation, new methods for evaluating long-form generations in this regime, and analyzing the performance of neural generation models on this task.

1.1 Thesis Overview and Contributions

In Chapter 2 we review standard approaches to structured prediction in NLP, as well as how neural network methods have affected these approaches. In Chapter 3 we tackle the task of coreference resolution both with and without the benefit of a search-based, structured approach. In Chapter 4 we show how to recast the ubiquitous sequence-to-sequence model (Sutskever et al., 2011, 2014) in the framework of search-based structured prediction. Finally, in Chapter 5 we address the structured prediction problem of long-form text generation. In summary, this thesis makes the following contributions:

- A simple, neural model for coreference resolution, which attained state-of-the-art performance on its introduction (Chapter 3)
- A structured, neural model for coreference resolution, which further improved the state of the art (Chapter 3)
- A modified sequence-to-sequence model and associated training scheme, which addresses several issues associated with the standard sequence-to-sequence model and the way it is typically trained (Chapter 4)
- A new dataset for database-to-document generation, a new approach to evaluating systems on this task, and an analysis of the challenges posed by this task for sequence-to-sequence generation models (Chapter 5)

2

Background

2.1 Prediction Problems in NLP

Much work in modern Natural Language Processing is concerned with utilizing machine learning techniques for the purpose of making predictions related to linguistic phenomena. We will refer to scenarios where a particular sort of prediction is desired in response to a particular sort of input as "prediction problems." Prediction problems come with a set $\mathcal{X}$ of candidate inputs and a set $\mathcal{Y}$ of candidate outputs; $\mathcal{Y}$ contains all the valid predictions for any input $x \in \mathcal{X}$. For example, for a sentiment classification problem, $\mathcal{X}$ might be the set of English sentences, and $\mathcal{Y}$ the set of sentiment-labels (e.g., *positive*, *negative*, *neutral*) that might be applied to an English sentence.

The standard approach to making predictions in the context of a given prediction problem is to define a function

$$\text{score} : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}, \tag{2.1}$$

which gives a real score to a pairing of an input $x \in \mathcal{X}$ and a candidate label $y \in \mathcal{Y}$, and to predict

$$\hat{y} = \operatorname*{arg\,max}_{y' \in \mathcal{Y}} \text{score}(x, y'), \tag{2.2}$$

the highest scoring candidate in $\mathcal{Y}$. It is typical to parameterize the score function in terms of a set of parameters θ, and we will write score_{θ} when we wish to emphasize the dependence on θ. As an example of this general setup, in the sentiment-labeling problem above, given a sentence x, score would associate a real score with each of the three possible labels for x, and we would then predict the highest scoring label. Problems such as this, where we attempt to predict one of V discrete labels for an input, are known as multi-class classification problems.

Making predictions using the prediction rule in Equation (2.2), that is, by selecting the output $\hat{y}$ with the highest score, can be motivated in different ways, often depending on the form of the score function. For instance, if we let X be a random variable taking values in $\mathcal{X}$, and Y be a random variable taking values in $\mathcal{Y}$, we may take $\text{score}_{\theta}(x, y)$ to be $p_{\theta}(Y = y \,|\, X = x)$. We may then motivate Equation (2.2) from the perspective of statistical decision theory (Friedman et al., 2001; Smith, 2011, Chapters 2 and 3, resp.). In particular, we first formalize the badness of predicting $\hat{y}$ for an input x when the *true* output is y using a loss function $\ell : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}_{\geq 0}$. According to statistical decision theory, we ought to predict the $\hat{y}$ with the lowest expected loss, where the expectation is taken over the distribution $p_\theta(Y = y \mid X = x)$. If we moreover define the loss as $\ell(x, y, \hat{y}) = \mathbf{1}_{\{\hat{y} \neq y\}}$, we have

$$\begin{aligned}
\hat{y} &= \operatorname*{arg\,min}_{y' \in \mathcal{Y}} \sum_{y \in \mathcal{Y}} p_\theta(Y = y \mid X = x)\, \ell(x, y, y') \\
&= \operatorname*{arg\,min}_{y' \in \mathcal{Y}} \sum_{y \in \mathcal{Y}} p_\theta(Y = y \mid X = x)\, \mathbf{1}_{\{y' \neq y\}} \\
&= \operatorname*{arg\,max}_{y' \in \mathcal{Y}} \text{score}_\theta(x, y').
\end{aligned}$$

For non-probabilistic score functions, Equation (2.2) is often motivated from the perspective of statistical learning theory. For instance, it is possible to show that in multi-class classification scenarios the expected misclassification rate of making predictions as in Equation (2.2) is, with probability at least $1 - \delta$, upper-bounded by the rate at which a set of N *training* examples is either misclassified by Equation (2.2) or for which the score of $\hat{y}$ does not exceed the score of the second highest-scoring output by a positive margin ρ, plus some terms that depend on the expressiveness of score, N, and ρ (Mohri et al., 2012, Chapter 8).

2.1.1 Structured Prediction

Many prediction problems in natural language processing, such as tagging, parsing, coreference resolution, and machine translation, require the prediction of outputs that are *structured*. In these cases, for instance, we are interested in predicting a sequence of part-of-speech labels, a parse tree, a partition of a document's noun phrases, or a well-formed sentence, respectively. Intuitively, these sorts of predictions differ from those made in multi-class classification, in that the objects being predicted have internal structure (unlike the discrete labels we predict in multi-class classification problems). Consequently, the number of candidate outputs for a particular input x also tends to be exponential in the size of x. In an attempt to formalize the notion of a structured prediction problem, Daumé III (2006) offers two necessary (and jointly sufficient) conditions for a prediction problem to be considered a structured prediction problem. They are

(1) that each $y \in \mathcal{Y}$ be encodable as at least one element of $\mathcal{V}^{T_y}$, where $\mathcal{V} = \{1, \ldots, V\}$, and where V and each T_y is a positive integer, with T_y possibly depending on y.

(2) that there is no polynomially sized encoding of elements of $\mathcal{Y}$ such that ℓ decomposes component-wise over the encoding.

As Daumé III (2006) notes, Condition (1) is quite general, and includes intuitively unstructured prediction problems such as those involving mere multi-class classification. Condition (2) attempts to rectify this by guaranteeing that the loss does not decompose as the sum of component-wise losses over each component in the encoding, making it inadvisable to predict each component of the encoding independently. At the same time, Daumé III (2006) himself notes that Condition (2) may be too strong, as it precludes classical structured prediction problems, such as sequence labeling, under standard losses such as Hamming Loss, which decomposes as the sum of per-label losses. Accordingly, it is standard to operate with an intuitive notion of what constitutes a structured prediction problem, with the relevant intuition in particular being that the score function somehow takes into account the interdependence between the components or parts of each $y \in \mathcal{Y}$; we elaborate more on this point below.

2.1.2 Structured Prediction Preliminaries

Approaches to supervised structured prediction differ from each other along three principal axes, namely, the form of the score function, the way the $\arg\max$ in Equation (2.2) is computed or approximated, and the loss function used to learn the parameters θ, which parameterize score. Before delving into each of these approaches, we provide additional background here on how the score function is typically parameterized, and how these parameters are learned.

Features The score function in Equation (2.1) is typically parameterized in terms of some features of an input x and candidate output y, as well as real weights associated with each of these features. Intuitively, we would like our features to capture information relevant to determining whether y is a good prediction for x. Formally, let $\boldsymbol{\phi} : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^D$ be a function mapping an input $x \in \mathcal{X}$ and a candidate output $y \in \mathcal{Y}$ into a real, D-dimensional vector. We may then calculate score(x, y) as $\boldsymbol{w}^{\mathsf{T}}\boldsymbol{\phi}(x, y)$, where $\boldsymbol{w} \in \mathbb{R}^D$ and $\boldsymbol{w} \in \theta$.[1] In classical NLP systems, $\boldsymbol{\phi}$ picked out binary predicates that held (or didn't) of an x, y pair. For instance, in scoring a candidate translation y for an input x, we might wish to have $\boldsymbol{\phi}(x, y)_1 = \mathbf{1}_{\{\text{len}(x) = \text{len}(y)\}}$, where $\mathbf{1}_{\{q\}}$ returns 1 if q is true, and 0 otherwise, and where len returns the length of a sequence. As we will see in the next section, neural NLP views $\boldsymbol{\phi}$ itself as a parameterizable function of x and y.

Decomposing Elements of $\mathcal{Y}$ It is typical to assume that the candidate structures in $\mathcal{Y}$ are composed of more elementary sub-structures or "parts." For instance, the elementary parts of a sequence of labels might be the labels themselves; if $\mathcal{Y}$ contains trees, we might view each tree in $\mathcal{Y}$ as being composed of elementary sub-trees of depth 1, or of directed edges. Note that in Condition (1) above we insisted that each element in $\mathcal{Y}$ be encodable as a T-length sequence over $\mathcal{V}$, and so for simplicity we will identify elements of $\mathcal{V}$ with the parts

[1] Note that for a probabilistic score, this linear score will be used to form a normalized distribution.

to which they correspond, and assume that each y has T parts.[2] This decomposition of each y into parts is often reflected explicitly in the features, which examine only a subset of y's parts at a time. For instance, if $\mathcal{Y}$ contains sequences of labels (with each label in $\mathcal{V}$), we might decide to model interactions only between adjacent labels, which gives rise to score functions such as $\boldsymbol{w}^{\mathsf{T}}\boldsymbol{\phi}(x,y) = \sum_{t=1}^{T} \boldsymbol{w}_t^{\mathsf{T}}\boldsymbol{\phi}_t(x, y_t, y_{t-1})$. If we view x and y as instantiations of random variables, then we can identify the subsets of y's parts examined by the $\boldsymbol{\phi}_t$ as the maximal cliques in an undirected graphical model (and the associated scores with their corresponding potentials). To de-clutter notation, we will also occasionally write $\boldsymbol{\phi}_t(x,y)$ – that is, without subscripting y – to indicate that $\boldsymbol{\phi}_t$ are features used in predicting the t'th part of y, without specifying which other parts of y are examined by $\boldsymbol{\phi}_t$.

Learning Parameters θ are learned by minimizing a loss function defined on a set of training examples $\{x^{(n)}, y^{(n)}\}_{n=1}^{N}$, where $y^{(n)}$ is the true or "gold" structure for input $x^{(n)}$. Loss functions used for training take the following form (plus, perhaps, a term penalizing the model complexity of score):

$$L(\theta) = \sum_{n=1}^{N} \ell(x^{(n)}, y^{(n)}, \hat{y}^{(n)}),$$

where we have written $L(\theta)$ to emphasize that the local loss terms ℓ interact with $x^{(n)}$, $y^{(n)}$, and $\hat{y}^{(n)}$ through the function score_θ, and that training consists of minimizing L with respect to θ; we typically also assume that L is at least sub-differentiable with respect to θ. Examples of specific L and ℓ functions will be given in what follows.

[2] As we note in Condition (1), T may depend on y, though we will leave this implicit in order to keep the notation simple.

2.2 Structured Prediction when Inference is Tractable

We begin by discussing approaches to structured prediction where the computation of the $\arg\max$ in Equation (2.2) is tractable.

2.2.1 Maximum Entropy Markov Models (MEMMs)

MEMMs (McCallum et al., 2000) define a probabilistic model of sequences $y \in \mathcal{Y} = \mathcal{V}^T$ given an input sequence x, also of length T. The probability of a sequence is modeled as the product of locally normalized per-label distributions, as follows:

$$\text{score}(x, y) = p(y \,|\, x) = \prod_{t=1}^{T} p(y_t \,|\, x_t, y_{t-1}) = \prod_{t=1}^{T} \frac{\exp\left(\boldsymbol{w}_t^{\mathsf{T}} \boldsymbol{\phi}_t(x_t, y_t, y_{t-1})\right)}{\sum_{v \in \mathcal{V}} \exp\left(\boldsymbol{w}_t^{\mathsf{T}} \boldsymbol{\phi}_t(x_t, v, y_{t-1})\right)}. \tag{2.3}$$

Note that the above model makes a first-order Markov assumption; that is, it assumes that, conditioned on y_{t-1}, the label y_t is independent of any y_j preceding y_{t-1}. This independence assumption manifests itself in the form of the features used above, which only depend on x_t, y_t, and y_{t-1} for each t.[3] This independence assumption also allows for the $\arg\max$ sequence to be calculated exactly with a Viterbi-style dynamic programming algorithm, almost identical to that used in doing inference in Hidden Markov Models (Rabiner, 1989).

Given a dataset $\{x^{(n)}, y^{(n)}\}_{n=1}^{N}$ of input sequences paired with their true output sequences, MEMMs are trained by minimizing the negative log-likelihood of the $y^{(n)}$, giving rise to the

[3] While MEMM features are often presented as conditioning only on x_t, they may additionally condition on the rest of x without hurting tractability.

following loss function:

$$
\begin{aligned}
L(\theta) &= -\sum_{n=1}^{N}\sum_{t=1}^{T} \ln p(y_t^{(n)} \,|\, x_t^{(n)}, y_{t-1}^{(n)}) \\
&= -\sum_{n=1}^{N}\sum_{t=1}^{T} \boldsymbol{w}_t^{\mathsf{T}} \boldsymbol{\phi}_t(x_t^{(n)}, y_t^{(n)}, y_{t-1}^{(n)}) - \ln \sum_{v \in \mathcal{V}} \exp\left(\boldsymbol{w}_t^{\mathsf{T}} \boldsymbol{\phi}_t(x_t^{(n)}, v, y_{t-1}^{(n)})\right). \qquad (2.4)
\end{aligned}
$$

MEMMs are known to have two fundamental drawbacks, one relating to the loss used in their training, and one relating to the model itself (that is, to how the score is computed under an MEMM). These are elaborated upon below.

Issues with MEMM Training In terms of the loss (2.4), note that the loss term associated with the t'th label, $\ln p(y_t^{(n)} \,|\, x_t^{(n)}, y_{t-1}^{(n)})$, conditions on the *true* $y_{t-1}^{(n)}$. That is, MEMMs are trained to predict the t'th label of $y^{(n)}$ conditioned on the *true* previous label. When making predictions at test time, however, we will not have access to true previous labels; we must use our predicted (probabilities for) y_{t-1} in calculating the $\arg\max$. This mismatch between the way the model is trained and its use at test time is now referred to as *exposure bias* (Ranzato et al., 2016), since the model is not exposed to its own predictions at training time, but was recognized as a pathology much earlier (Lafferty et al., 2001; Daumé III, 2006; Kääriäinen, 2006).

Issues with the MEMM Model As is evident from Equation (2.3), the probability of a sequence y for an input x is the product of locally-normalized per-label probabilities. This leads to a pathology known as *label bias* (Bottou, 1991; Lafferty et al., 2001), where all the mass associated with $p(y_t = v_0 \,|\, y_{1:t-1}, x)$ must be consumed by the probabilities that condition on $y_t = v_0$. As a result, even if *no* predictions $v \in \mathcal{V}$ look appropriate at time $t+1$, we can at best distribute the mass uniformly among $\mathcal{V}$, rather than give any predictions emanating from our choice of v_0 at time t zero probability, as we might wish. This observation also makes it clear that this local normalization property limits the expressiveness of the conditional distributions that can be formed, and indeed it can be shown (Smith and Johnson, 2007) that the set of conditional distributions expressible by models such as that in Equation (2.3) form a strict subset of those expressible by globally normalized models, such as HMMs or CRFs, which are introduced in the next subsection.

2.2.2 Conditional Random Fields (CRFs)

A CRF (Lafferty et al., 2001) specifies the conditional distribution of a random variable Y given a random variable X with respect to the factorization implied by an undirected graphical model over X, Y. In the case of a conditional sequence distribution with a first-order Markov property (as in the MEMM case), this distribution may be written:

$$\text{score}(x, y) = p(y \,|\, x) = \frac{1}{Z(x)} \prod_{t=1}^{T} \exp\left(\boldsymbol{w}_t^{\mathsf{T}} \boldsymbol{\phi}_t(x, y_t, y_{t-1})\right), \tag{2.5}$$

where

$$Z(x) = \sum_{y' \in \mathcal{Y}} \prod_{t=1}^{T} \exp\left(\boldsymbol{w}_t^{\mathsf{T}} \boldsymbol{\phi}_t(x, y'_t, y'_{t-1})\right). \tag{2.6}$$

$Z(x)$ is known as the "partition function," and it guarantees that the conditional probabilities sum to 1. Note that, unlike in the MEMM case, the distribution $p(y \,|\, x)$ is not the product of locally normalized distributions; it is rather globally normalized, using the partition function, and so this model does not suffer from label bias.

Given the Markov assumptions implicit in the model above, we may again find the arg max sequence exactly, using a Viterbi-style algorithm. Training minimizes the negative log-likelihood:

$$L(\theta) = -\sum_{n=1}^{N} \sum_{t=1}^{T} \boldsymbol{w}_t^{\mathsf{T}} \boldsymbol{\phi}_t(x^{(n)}, y_t^{(n)}, y_{t-1}^{(n)}) - \ln Z(x^{(n)}). \tag{2.7}$$

Note that, unlike in Equation (2.4), the loss above does not condition on a true history of predictions. (This can be seen in that the partition function does not make reference to the gold y, unlike the normalizing terms in (2.4)). Thus, CRFs additionally do not suffer from exposure bias.

It is typical to minimize Equation (2.7) with a gradient-based optimization algorithm. In order to do so, we must at least be able to efficiently calculate both the partition function $Z(x)$ as well as the gradient of $Z(x)$ with respect to the $\boldsymbol{w}_t$. This latter quantity turns out to involve an expectation with respect to the distribution of label-pairs y_t, y_{t-1} for each t, and so we must be able to compute the marginals $p(y_t, y_{t-1} \,|\, x)$ efficiently. In the case of a sequence model with first-order Markov properties (known as “linear-chain CRFs”), all of this can be done efficiently with dynamic programming.

While we have only explicitly presented linear-chain CRFs, CRFs may be applied to the prediction of y’s with arbitrary graphical model structure, where in particular we associate clique potentials with features $\boldsymbol{\phi}_t$ that examine the t’th clique in the graph. Recalling our notation from Section 2.1.2 for specifying features which examine only a subset of y’s parts at a time, we can write the score under a general CRF as

$$\text{score}(x, y) = p(y \,|\, x) = \frac{1}{Z(x)} \prod_{t=1}^{T} \exp\left(\boldsymbol{w}_t^{\mathsf{T}} \boldsymbol{\phi}_t(x, y)\right), \tag{2.8}$$

where we have again assumed y has T parts. In the case of general CRFs, there may no longer be efficient algorithms for computing the $\arg\max$, or for computing $Z(x)$ or the clique marginals; see Sutton et al. (2012) for additional details on both linear-chain and general CRFs.

2.2.3 Structured Perceptron and Structured SVM

Whereas structured prediction with CRFs requires the ability to efficiently compute the $\arg\max$ of the score function, the partition function, and the clique marginals, Structured Perceptron (Collins, 2002) and certain Structured SVM (Taskar et al., 2004; Tsochantaridis et al., 2005) approaches require only that we be able to efficiently compute the $\arg\max$. These latter approaches are non-probabilistic, and they utilize a simple form for the score:

$$\text{score}(x, y) = \sum_{t=1}^{T} \boldsymbol{w}_t^{\mathsf{T}} \boldsymbol{\phi}_t(x, y). \tag{2.9}$$

There are two, similar loss functions associated with this family of approaches. To define them, we first define a cost function $\Delta : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}_{\geq 0}$, which allows us to further customize the cost of predicting $\hat{y}$ for x when the true answer is y. The first loss function, known as a "margin rescaling" loss, and implied by a Quadratic Program originally proposed by Taskar et al. (2004), is defined as

$$L(\theta) = \sum_{n=1}^{N} \max_{y' \in \mathcal{Y}/y^{(n)}} \Big[\Delta(x^{(n)}, y^{(n)}, y') + \text{score}(x^{(n)}, y') - \text{score}(x^{(n)}, y^{(n)})\Big]_+ \tag{2.10}$$

where the set $\mathcal{Y}/y^{(n)}$ is equal to $\mathcal{Y}$ with $y^{(n)}$ removed, and the notation $[r]_+$ denotes $\max\{0, r\}$. The Structured Perceptron loss can be obtained from the loss above by having Δ always return 0.

The second loss, known as a "slack rescaling" loss, and implied by a Quadratic Program originally proposed by Tsochantaridis et al. (2005), is defined as

$$L(\theta) = \sum_{n=1}^{N} \max_{y' \in \mathcal{Y}/y^{(n)}} \Big[\Delta(x^{(n)}, y^{(n)}, y') \Big(1 + \text{score}(x^{(n)}, y') - \text{score}(x^{(n)}, y^{(n)})\Big)\Big]_{+} . \tag{2.11}$$

While these (non-Perceptron) losses were originally proposed as Quadratic Programs, often along with custom minimization algorithms, it is also possible to minimize them with stochastic, subgradient-descent based methods (Shalev-Shwartz et al., 2011). In particular, we may minimize the above losses by iteratively finding,

$$\hat{y}^{(n)} = \operatorname*{arg\,max}_{y' \in \mathcal{Y}/y^{(n)}} \Delta(x^{(n)}, y^{(n)}, y') + \text{score}(x^{(n)}, y') - \text{score}(x^{(n)}, y^{(n)}) \tag{2.12}$$

if we are minimizing Loss (2.10), or

$$\hat{y}^{(n)} = \operatorname*{arg\,max}_{y' \in \mathcal{Y}/y^{(n)}} \Delta(x^{(n)}, y^{(n)}, y') \Big(1 + \text{score}(x^{(n)}, y') - \text{score}(x^{(n)}, y^{(n)})\Big) , \tag{2.13}$$

if we are minimizing Loss (2.11), and then taking a gradient step to minimize the quantity inside the square brackets of Losses (2.10) and (2.11), respectively, if it is positive. Algorithm 1 illustrates using online, stochastic subgradient descent to minimize the margin rescaled objective; it may be modified simply to operate in mini-batches. Note that the maximizations in (2.12) and (2.13) are slightly different than the standard $\arg\max$ in Equation (2.2), due to the Δ term. This maximization, known as "loss-augmented inference," may be intractable to compute for some definitions of Δ, with the maximization in (2.13) in particular often being intractable due to the multiplicative term.

Algorithm 1 Online, margin rescaled SSVM training

```
procedure OnlineSSVM($\{x^{(n)}, y^{(n)}\}_{n=1}^{N}, \eta, \text{epoch}_{\text{max}}$)
    Init parameters $\theta$
    for epoch $= 1, \ldots, \text{epoch}_{\text{max}}$ do
        for $n = 1, \ldots, N$ do
            $\hat{y}^{(n)} = \arg\max_{y' \in \mathcal{Y}/y^{(n)}} \Delta(x^{(n)}, y^{(n)}, y') + \text{score}(x^{(n)}, y') - \text{score}(x^{(n)}, y^{(n)})$
            if $\Delta(x^{(n)}, y^{(n)}, \hat{y}^{(n)}) + \text{score}(x^{(n)}, \hat{y}^{(n)}) - \text{score}(x^{(n)}, y^{(n)}) > 0$ then
                $\theta \leftarrow \theta - \eta \frac{\partial}{\partial \theta} \left[\Delta(x^{(n)}, y^{(n)}, \hat{y}^{(n)}) + \text{score}(x^{(n)}, \hat{y}^{(n)}) - \text{score}(x^{(n)}, y^{(n)})\right]$
    return $\theta$
```

Finally, we note that the Structured Perceptron and SSVMs suffer from neither exposure bias nor label bias, since they do not assume a gold history during training, and they do not calculate locally normalized scores. Furthermore, the Δ terms theoretically allow a loss that does not decompose over parts of $\hat{y}$, though, as noted above, a non-decomposable loss might make calculating the $\arg\max$ intractable.

2.3 Structured Prediction when Inference is Intractable

All of the approaches described in the previous section may be modified to work in situations where the $\arg\max$ (and clique marginals and the partition function, in the case of CRFs) cannot be tractably computed, by approximating the desired quantities; see Ratnaparkhi (1996), Sutton et al. (2007), and Finley and Joachims (2008), for examples of approximating inference within MEMM-style modeling, CRFs, and SSVMs, respectively. However, there are additional structured prediction approaches that are trained to take into account the fact that the $\arg\max$ must be approximated. Here in particular we will review search-based approaches to structured prediction.

2.3.1 Approximate Search

While the dynamic programming algorithms for calculating the $\arg\max$ discussed in the previous section are also search algorithms, here we will use the term "search" to refer to approximate algorithms. These approximate search algorithms heuristically search for the $\arg\max$, but are not guaranteed to find it. Search algorithms for structured prediction assume, as we did in Section 2.1.2, that the structures in $\mathcal{Y}$ decompose into sequences of "parts." These algorithms then incrementally construct $\hat{y}$ by searching for the t'th part of $\hat{y}$, which we will denote by $\hat{y}_t$, given the history of incremental predictions preceding the t'th part, which we will denote by $\hat{y}_{1:t-1}$.[4]

The most popular inexact search algorithm for structured prediction is *beam search*, which assumes access to four ingredients:

1. a function succ mapping x and a partial structure $\hat{y}_{1:t-1}$ with $t-1$ parts to the set of all valid structures for x with t parts;
2. a designated "start" partial structure, $\hat{y}_0$;
3. a way to determine whether a partial structure is complete;
4. a function $f : \mathcal{X} \times \mathcal{V}^* \rightarrow \mathbb{R}$ that scores the quality of a partial structure $\hat{y}_{1:t}$ for an input x. f should be chosen such that $\arg\max_{y' \in \mathcal{Y}} f(x, y') = \arg\max_{y' \in \mathcal{Y}} \text{score}(x, y')$.

Beam search operates by maintaining a "beam" – that is, an ordered list – of K partial structures $\hat{y}^{(1:K)}_{1:t-1}$, where $\hat{y}^{(k)}_{1:t-1}$ is the k'th highest scoring partial structure on the beam. At each round, all the partial structures of length t that can be formed from partial structures on the beam via the succ function are scored with f, and the top K are selected to form the beam for the next round. Search terminates either after a pre-defined number of rounds or after there are no remaining hypotheses. Pseudocode for beam search is show in Algorithm 2.

[4] The history of incremental predictions may be identified with a "state" in the classical AI search literature (Russell et al., 2003).

Algorithm 2 Beam Search for finding an approximate arg max

```
procedure BeamSearch($x, K$)
    $\hat{y} \leftarrow$ null
    Init beam $\hat{y}_{1:0}^{(1:K)}$ with $K$ copies of $\hat{y}_0$
    for $t = 1, \ldots, T_{\max}$ do
        $\hat{y}_{1:t}^{(1:K)} \leftarrow \mathrm{topK}_f(\bigcup_{k=1}^{K} \mathrm{succ}(x, \hat{y}_{1:t-1}^{(k)}))$
        if $\hat{y}_{1:t}^{(1:K)}$ is empty then
            break
        for $k = 1, \ldots, K$ do
            if $\hat{y}_{1:t}^{(k)}$ is finished and $f(x, \hat{y}_{1:t}^{(k)}) > f(x, \hat{y})$ then
                $\hat{y} \leftarrow \hat{y}_{1:t}^{(k)}$
  return $\hat{y}$
```

There, we use the notation topK_f to denote an operator that selects the K partial structures from a set that are scored highest by the scoring function f.

While beam search can be used to approximate the arg max for the non-probabilistic structured prediction approaches discussed in the previous section, in the next subsection we review structured prediction approaches which are aware, at training time, of the fact that beam search is being used.

2.3.2 Incremental Perceptron (IP) and LaSO

The Incremental Perceptron (Collins and Roark, 2004) uses a similar model to that used in the Structured Perceptron, except scores are now given to partial structures:

$$f(x, y_{1:t}) = \sum_{j=1}^{t} \boldsymbol{w}_j^{\mathsf{T}} \boldsymbol{\phi}_j(x, y), \tag{2.14}$$

as is required for beam search. In addition, IP performs beam search during training time. In order to define the IP loss, we will say $y^{(n)}$ is "reachable" from some partial structure $\hat{y}_{1:t}$ if there is a sequence of partial structures $\hat{y}_{1:t+i}$ for $i = 1, \ldots, T-t$ such that $\hat{y}_{1:t+i} \in \mathrm{topK}_f\, \mathrm{succ}(x, \hat{y}_{1:t+i-1})$, and such that $\hat{y}_{t:t+T-t} = y^{(n)}$. We will say $y^{(n)}$ is "unreachable" from some partial structure $\hat{y}_{1:t}$ if it is not reachable.

With that definition in hand, we can understand IP as using the same loss as does the Structured Perceptron, except for two differences: first, Line 4 in Algorithm 1 is approximated with beam search. Second, if at any point t during beam search $y^{(n)}$ becomes unreachable from *all* the $\hat{y}^{(k)}_{1:t}$, then θ is immediately updated as

$$\theta \leftarrow \theta - \eta \frac{\partial}{\partial \theta} \Delta(x^{(n)}, y^{(n)}_{1:t}, \hat{y}^{(n,1)}_{1:t}) + f(x^{(n)}, \hat{y}^{(n,1)}_{1:t}) - \text{score}(x^{(n)}, y^{(n)}_{1:t}), \tag{2.15}$$

where we use the notation $\hat{y}^{(n,1)}_{1:t}$ to denote the first (and thus highest scoring) partial structure of size t on the beam. The remainder of the n'th example is ignored, and training continues from the next example. Because training updates as soon as $y^{(n)}$ becomes unreachable, and then moves on to the next example, IP is also often referred to as training with "early-update."

LaSO (Daumé III and Marcu, 2005), short for "Learning as Search Optimization," is similar to IP, except training on the n'th example is *not* aborted if $y^{(n)}$ becomes unreachable from all the $\hat{y}^{(k)}_{1:t}$ at some point t during beam search. Instead, the parameters θ are updated, and search resumes by resetting the beam at time $t+1$ to $\text{succ}(x^{(n)}, y^{(n)}_{1:t})$, that is, to successors of the *gold* partial structure of size t. In LaSO, the θ update is also slightly different than that in Equation (2.15). In particular, it is either

$$\theta \leftarrow \theta - \eta \frac{\partial}{\partial \theta} \left[\left(\sum_{k=1}^{K} \Delta(x^{(n)}, y^{(n)}_{1:t}, \hat{y}^{(n,k)}_{1:t}) + f(x^{(n)}, \hat{y}^{(n,k)}_{1:t}) \right) - \text{score}(x^{(n)}, y^{(n)}_{1:t}) \right], \tag{2.16}$$

or a slight variant of the above based on the ALMA loss of Gentile (2001).

The important thing to note about IP and LaSO (as well as more recent generalizations (Huang et al., 2012)) is that these approaches attempt to correct for the inexact search to be used at test time by encouraging the model to do a better job at searching during training time. This is accomplished by scoring *partial* structures, rather than only full structures, and by explicitly training the model to avoid finding partial structures from which $y^{(n)}$ becomes unreachable. We also note that this family of methods addresses both exposure bias, since training does *not* assume the gold history of incremental predictions, as well as label bias, assuming the scores are not locally normalized. Furthermore, it can address non-decomposable losses in the same way as structured SVM.

2.3.3 SEARN, DAgger, AggreVaTe, and LOLS

Whereas IP and LaSO run search during training, in this subsection we review approaches that train a multi-class classifier to predict the next part of $\hat{y}$ to add to $\hat{y}_{1:t-1}$ in a way that both avoids exposure bias and allows for non-decomposable losses, but without explicitly searching at training time. This family of approaches is exemplified by SEARN (Daumé III et al., 2009), DAgger (Ross et al., 2011), AggreVaTe (Ross and Bagnell, 2014), and LOLS (Chang et al., 2015), and they are taken to offer a sound approach to predicting structures in a greedy way, though we discuss whether these approaches must truly be greedy in the next section.

These algorithms have close connections to imitation learning, where a “policy” is learned to predict the next part of $\hat{y}^{(n)}$ by being given access to an expert demonstration, namely, $y^{(n)}$. Indeed, whereas the above algorithms are typically presented using Markov Decision Process (MDP) terminology, we will attempt to avoid much of this machinery in our presentation in order to minimize the introduction of new notation. The one new concept we will need

Algorithm 3 Imitation Learning-style L2S

```
procedure ImitationL2S($\{x^{(n)}, y^{(n)}\}_{n=1}^{N}, \pi_{\text{ref}}, \text{epoch}_{\text{max}}$)
    $\pi \leftarrow \pi_{\text{ref}}$; $\Gamma \leftarrow \emptyset$
    for epoch $= 1, \ldots, \text{epoch}_{\text{max}}$ do
        if SEARN then
            $\Gamma \leftarrow \emptyset$
        for $n = 1, \ldots, N$ do
            for $t = 1, \ldots, T$ do
                Let $\pi_{\text{in}}$ be the current roll-in policy        ▷ Depends on algorithm
                $\hat{y}_{1:t} \leftarrow \pi_{\text{in}}^{t}(\hat{y}_0)$
                $\boldsymbol{c} \leftarrow \boldsymbol{0}$        ▷ Init costs for each next-part
                for $v \in \mathcal{V}$ do
                    Let $\pi_{\text{out}}$ be the current roll-out policy        ▷ Depends on algorithm
                    $\hat{y}_{1:t,v,t+2:T} \leftarrow \pi_{\text{out}}^{T-t-1}(\hat{y}_{1:t,v})$
                    $c_v \leftarrow \ell(x^{(n)}, y^{(n)}, \hat{y}_{1:t,v,t+2:T})$
                $\boldsymbol{c} \leftarrow \boldsymbol{c} - \min_v \boldsymbol{c}$
                Add cost-sensitive example $(\boldsymbol{\phi}_t(x^{(n)}, \hat{y}_{1:t}), \boldsymbol{c})$ to $\Gamma$
            if LOLS or AggreVaTe then
                Update $\pi$ after training on $\Gamma$; $\Gamma \leftarrow \emptyset$        ▷ Depends on algorithm
        if SEARN or DAgger then
            Update $\pi$ after training on $\Gamma$; $\Gamma \leftarrow \emptyset$        ▷ Depends on algorithm
    return $\pi$
```

is that of a "policy" π, which we will take to be a cost-sensitive, multi-class classifier, which maps an input x and a partial structure $\hat{y}_{1:t}$ to a distribution over next-parts $v \in \mathcal{V}$; thus, by sampling a next-part v from the policy distribution, we can form $\hat{y}_{1:t+1}$ from $\hat{y}_{1:t}$. For example, in the case of predicting a sequence of labels, π would provide a distribution over labels for the $t + 1$'st label in the sequence, given x and a current sequence of labels of length t. These algorithms train the multi-class classifier π by iteratively "rolling in" – that is, making sequential predictions using the learned π – to a partial structure $\hat{y}_{1:t}$, and then providing costs to π for each potential next-part prediction v derived by "rolling out" from the partial structure $\hat{y}_{1:t,v}$ to a complete structure $\hat{y}_{1:T}$. Note that we have used the notation $\hat{y}_{1:t,v}$ to refer to the partial structure with $t+1$ parts formed by adding v as the $t+1$'st part to $\hat{y}_{1:t}$.

In Algorithm 3, we present relatively generic pseudo-code encompassing training with SEARN, DAgger, AggreVaTe, and LOLS, inspired by a similar presentation of the first two of these algorithms in Vlachos (2013). There, we use the notation $\pi^j(x, \hat{y}_{1:t})$ to denote the result of recursively sampling next-parts from π to form a partial structure $\hat{y}_{1:t+j}$.

SEARN, DAgger, AggreVaTe, and LOLS assume access to a reference policy π_{ref}. The first three of these algorithms require that π_{ref} be optimal in the sense that π_{ref} chooses the lowest-loss action assuming all subsequent choices are made optimally. LOLS, on the other hand, allows π_{ref} to be suboptimal. These algorithms then proceed by rolling-in for T steps with a roll-in policy π_{in}, to form T partial structures; see Line 8. LOLS and AggreVaTe roll-in with the current, learned π, whereas SEARN and DAgger roll-in with a mixture of π and π_{ref}.[5] Then, for each partial structure, the cost $\boldsymbol{c} \in \mathbb{R}^V$ of predicting each next-part v is calculated as $c_v = \ell(x^{(n)}, y^{(n)}, \hat{y}_{1:t,v,t+2:T}) - \min_{v'} \ell(x^{(n)}, y^{(n)}, \hat{y}_{1:t,v',t+2:T})$, where the parts following v are obtained using a roll-out policy π_{out}; see Line 12. SEARN and LOLS use a mixture of π and π_{ref} to roll-out, whereas DAgger and AggreVaTe use π_{ref}. The T partial structures and their associated costs $\boldsymbol{c}_t$ are then added to a dataset on which a cost-sensitive multi-class classifier may be trained. There are small differences here in how examples are processed, with LOLS being online, DAgger and AggreVaTe aggregating examples over the entire training process, and SEARN aggregating examples just over an epoch, as well as how π is updated after each batch of examples. At the end of training, we have a learned policy π, which we can use to incrementally construct structures for unseen inputs. Thus, in this framework, structured prediction amounts to iteratively applying a multi-class classifier, rather than performing explicit search.

[5] Also, SEARN samples a new mixture for each step t in the roll-in and the roll-out, unlike the other algorithms, which sample a policy for an entire example.

2.3.4 Features for Search-based Structured Prediction

Crucial to both IP/LaSO and SEARN/DAgger/AggreVaTe/LOLS techniques is the ability to score next-part predictions conditioned on the partial structure $\hat{y}_{1:t-1}$. Thus, some care is required to define features that allow the partial structure scorer f or the multi-class classifier π to accurately condition on this history of predictions. This can be difficult to accomplish with discrete, binary features, especially if $\hat{y}$ has many parts, and several authors go into detail about strategies for defining these features (Daumé III et al., 2009; Doppa et al., 2014). Unfortunately, because of the potential combinatorial explosion of discrete features targeting variable-length partial structures $\hat{y}_{1:t-1}$, in practice many feature sets for search-based structured prediction end up making Markov assumptions about the partial structures, even though (at least in theory) part of the power of this search-based framework is that it requires no Markov assumptions. We will see in the next section, however, that recurrent neural networks provide a simple solution to this issue, by learning a compact embedding of variable length histories that makes no Markov assumptions.

2.3.5 Discussion

Both IP/LaSO and SEARN/DAgger/AggreVaTe/LOLS rely on scoring next-parts in incrementally constructing a predicted structure. The major difference between the two approaches, then, is often taken to be that SEARN/DAgger/AggreVaTe/LOLS are employed in a greedy fashion,[6] whereas IP/LaSO allow for a slightly larger search space. In recent years, greedy

[6] Daumé III et al. (2009) notes that SEARN allows for beam search-style training by allowing the candidate next-parts of the structured prediction problem to be the beam of hypotheses formed by advancing *one* of the partial structures on a beam. One potential issue with this approach, however, is that we would still need a policy π_{ref} to specify the optimal way of advancing one partial structure on a beam, which may be difficult to specify.

inference has been embraced by deep learning practitioners, as have reinforcement-learning based approaches to sequential prediction, which *must* be greedy.

SEARN/DAgger/AggreVaTe/LOLS, then, which are quite close to being RL algorithms, are quite congenial to these recent trends. At the same time, as noted recently by Daumé III (2017), structured prediction is crucially different (and easier) than RL, in that the environment is not stochastic, and so it makes sense to search (at test time) for structures. That is, in structured prediction we can afford to consider multiple hypotheses (i.e., partial structures) as we search for the best-scoring overall structure. At test time in RL, of course, we can't consider multiple ways of flying a helicopter or steering a car at any particular moment; we *must* take an action at each time-step, after which the environment (possibly) changes. It is therefore somewhat surprising that deep-learning practitioners have largely been *training* their structured prediction models with RL algorithms in recent years. (Indeed, most of the structured prediction models trained with RL algorithms perform search at test time!). Thus, while RL-style training has certainly led to impressive empirical performance, it seems likely that further improvements might be made by taking search into account during training time as well.

2.4 Deep NLP Models

Here we take the view that the primary advantage of using neural networks in NLP is that they automatically *learn* useful and compact feature representations $\boldsymbol{\phi}_t(x, y)$ for inputs and candidate (partial) outputs, without having to specify these features manually. Below we briefly review some of the standard tools utilized in neural NLP.

2.4.1 Word and Feature Embeddings

Traditionally, the features $\boldsymbol{\phi}_t(x, y)$ used in NLP systems lived in $\{0, 1\}^D$, where D is the number of features. Features were manually defined to pick out binary properties of the input and (the t'th part of) y, and took on the value 1 if the property held, and 0 otherwise. The idea of word or feature embeddings is to associate a real vector, rather than a binary value, with every feature, and to learn this vector along with the rest of the parameters of the model (Bengio et al., 2003; Collobert et al., 2011). Thus, given a traditional feature vector $\boldsymbol{\phi}_t(x, y) \in \{0, 1\}^D$, we may obtain a feature embedding by taking $\boldsymbol{E}\,\boldsymbol{\phi}_t(x, y)$, where $\boldsymbol{E} \in \mathbb{R}^{D_{\text{emb}} \times D}$ is known as the matrix of embeddings.[7] In the case where $\boldsymbol{\phi}_t(x, y)$ is one-hot – that is, contains only a single 1 – $\boldsymbol{E}\,\boldsymbol{\phi}_t(x, y)$ simply retrieves the column in $\boldsymbol{E}$ corresponding to the non-zero feature, and when this non-zero feature represents a word, it is known as a word embedding.

Note that these embedded features may be used in all the models we have defined above, in place of their binary counterparts;[8] during training, $\boldsymbol{E}$ is updated using gradients $\nabla_{\boldsymbol{E}} L(\theta)$. Much empirical work in neural NLP has operated under the assumption that features derived from embeddings of simple binary features (such as binary word-level features) can match or outperform binary, hand-crafted features, a view espoused by Collobert et al. (2011).

[7] It is also common to apply a nonlinearity or concatenate columns of $\boldsymbol{E}$ in forming feature embeddings.

[8] This modification, though, comes at the cost of the loss no longer being convex in θ.

2.4.2 Multi-Layer Perceptrons

Multi-Layer Perceptrons (MLPs) compute a non-linear mapping from a vector $\boldsymbol{h}^i \in \mathbb{R}^{D_i}$ to another vector $\boldsymbol{h}^{i+1} \in \mathbb{R}^{D_{i+1}}$ as follows:

$$\boldsymbol{h}^{i+1} = g(\boldsymbol{W}^{i+1}\boldsymbol{h}^i + \boldsymbol{b}),$$

where $\boldsymbol{W}^{i+1} \in \mathbb{R}^{D_{i+1} \times D_i}$, $\boldsymbol{b} \in \mathbb{R}^{D_{i+1}}$, and g is a nonlinearity (such as tanh or ReLU (Nair and Hinton, 2010)) applied component-wise. Often, $\boldsymbol{h}^1 = \boldsymbol{\phi}_t(x, y)$, and so we can see MLPs as extracting a non-linear feature embedding from a feature (embedding) vector.

2.4.3 Convolutional Neural Networks

In NLP, convolutional neural networks (CNNs) (LeCun et al., 1998) are most often used to produce a shift-invariant feature representation of a sequence of feature embeddings, such as a sequence of word embeddings (Collobert et al., 2011; Kim, 2014). With $\boldsymbol{h}^i_1, \ldots, \boldsymbol{h}^i_T$ a sequence of feature embeddings in $\mathbb{R}^{D_i}$, the matrix $\boldsymbol{W} \in \mathbb{R}^{D_{i+1} \times K D_i}$ a matrix of D_{i+1} filters of width K (with $K < T$), and $\boldsymbol{b} \in \mathbb{R}^{D_{i+1}}$ a bias vector, we compute a sequence of $T - K + 1$ convolutional features in $\mathbb{R}^{D_{i+1}}$ as

$$\boldsymbol{h}^{i+1}_j = g(\boldsymbol{W}[\boldsymbol{h}^i_j, \ldots, \boldsymbol{h}^i_{j+K-1}] + \boldsymbol{b})$$

for $j = 1, \ldots, T - K + 1$, where the notation $[\boldsymbol{h}^i_j, \ldots, \boldsymbol{h}^i_{j+K-1}]$ denotes vector concatenation, and g is again a component-wise nonlinearity. We are often interested in obtaining a single feature representation from the sequence of convolutional features $\boldsymbol{h}^{i+1}_j$, and so it is common to extract a "pooled" representation, which takes the maximum or average value for each coordinate over the $T-K+1$ representations. For example, in the case of max-pooling, we obtain the pooled $\boldsymbol{h}^{i+1} \in \mathbb{R}^{D_{i+1}}$ with coordinates $h_d^{i+1} = \max\{h_{1,d}^{i+1}, \ldots, h_{T-K+1,d}^{i+1}\}$.

2.4.4 Recurrent Neural Networks

A recurrent neural network is a parameterized non-linear function, which we will always denote by **RNN**, that recursively maps a sequence of feature embedding vectors to a new sequence of vectors. Let $\boldsymbol{h}_1^i, \ldots, \boldsymbol{h}_T^i$ be a sequence of T vectors $\boldsymbol{h}_t^i \in \mathbb{R}^{D^i}$, and let $\boldsymbol{h}_0^i = \mathbf{0}$. Applying an RNN to any such sequence yields

$$\boldsymbol{h}_t^{i+1} \leftarrow \mathbf{RNN}(\boldsymbol{h}_t^i, \boldsymbol{h}_{t-1}^i; \theta),$$

where θ is the set of parameters for the model, which are shared over time, and each $\boldsymbol{h}_t^{i+1} \in \mathbb{R}^{D^{i+1}}$. There are several varieties of RNN, but the most commonly used in natural-language processing is the Long Short-Term Memory network (LSTM) (Hochreiter and Schmidhuber, 1997), particularly for language modeling (e.g., Zaremba et al. (2014)) and machine translation (e.g., Sutskever et al. (2014)).

In this thesis we will take the view that RNNs are especially useful and powerful for search-based structured prediction, because we may view the hidden state $\boldsymbol{h}_t^{i+1}$ at time t as a compact representation of the *history* of the sequence $\boldsymbol{h}_1^i, \ldots, \boldsymbol{h}_T^i$ through time t. Using RNNs in this way will allow search-based structured prediction algorithms to easily condition their incremental predictions on the history of incremental predictions made previously.

3

Neural Models for Coreference Resolution[1]

3.1 Introduction

In this chapter we introduce the structured prediction problem of coreference resolution. We then present two neural models, the first an unstructured, neural "mention-ranking" model, which attained state-of-the-art performance when it was first introduced. We then describe a neural, structured model, which builds upon this baseline mention-ranking model, and further improves performance. Consistent with the methodology propounded in this thesis, this

[1] The material in this chapter is adapted from Wiseman et al. (2015) and Wiseman et al. (2016b).

latter model decomposes the structured prediction problem into a sequence of incremental predictions, each of which conditions on all the previous predictions; the history of previous predictions is moreover efficiently represented with an RNN.

3.1.1 The Task of Coreference Resolution

Studies of when noun phrases corefer – that is, refer to the same entity – as well as methods for predicting coreference date back to early work in natural language understanding (Woods et al., 1972; Winograd, 1972), and computational discourse (Grosz, 1977; Webber, 1978; Hobbs, 1978). In its modern incarnation, the task of coreference resolution is fundamentally a clustering task. Given a document $x \in \mathcal{X}$ and the set $\mathcal{M}(x) = \{x_n\}_{n=1}^{N}$ of x's *mentions* – that is, syntactic units that can refer or be referred to – coreference resolution involves partitioning $\mathcal{M}(x)$ into a set of clusters $\{X^{(m)}\}_{m=1}^{M}$ such that all the mentions in any particular cluster $X^{(m)}$ refer to the same underlying entity. We show an example of a short document with annotated coreference information in Figure 3.1. There, co-clustered mentions have the same (non-black) color, whereas *singleton* mentions (i.e., mentions clustered with no other mentions) are in black. Since the mentions within a particular cluster may be ordered linearly by their appearance in the document,[2] we will use the notation x_n to refer to the n'th mention in a document x, and the notation $X_j^{(m)}$ to refer to the j'th mention in the m'th cluster.

As noted above, a valid clustering for the purposes of coreference resolution forms a partition of the mentions, placing each mention in exactly one cluster. Since there are most N partitioning clusters in a document of N mentions, we may represent a clustering with a sequence $y \in \mathcal{Y}$, where $\mathcal{Y} = \{1, \ldots, N\}^N$, and where $y_n = m$ if and only if x_n is a member of

[2] We assume nested mentions are ordered by their syntactic heads.

[Cadillac] posted a [3.2% increase] despite [new competition from [Lexus, the fledgling luxury-car division of [Toyota Motor Corp]]] . [[Lexus] sales] weren't available; [the cars] are imported and [Toyota] reports [[their] sales] only at [month-end] .

Figure 3.1: A passage from the CoNLL 2012 English Corpus (Pradhan et al., 2012) annotated with its coreference information. Mentions x_n are enclosed in square brackets; *singleton* mentions are in black font, and mentions in non-black font have the same color if and only if they are in the same coreference cluster $X^{(m)}$.

$X^{(m)}$. Coreference systems operate by predicting a clustering $\hat{y} \in \mathcal{Y}$, where $\hat{y}$ is the highest scoring clustering under some scoring function.

3.1.2 Approaches to Predicting Coreference

The vast majority of machine learning approaches to coreference resolution assume that a document's mentions $\mathcal{M}(x)$ are extracted automatically, using either a rule-based (Lee et al., 2011) or trained system (Ng and Cardie, 2002). Coreference resolution is almost always treated as a supervised learning task (though see Haghighi and Klein (2007)), and modern coreference datasets, such as those used in the CoNLL 2012 Shared Task (Pradhan et al., 2012), provide a "gold" (i.e., true) coreference clustering y for each document. In what follows we briefly outline the major approaches that have been used historically to predict coreference.

Mention-Pair Models Early machine learning approaches to coreference resolution sought to treat the task as an unstructured prediction problem, by making a binary coreference prediction for each pair of mentions x_j, x_k in a document. This becomes a supervised binary prediction problem by taking the label of the pair x_j, x_k to be 1 if x_j, x_k are co-clustered in the associated gold clustering y, and 0 otherwise. At test time, a predicted clustering $\hat{y}$ is formed by post-processing these binary, pairwise predictions (Soon et al., 2001; Ng and Cardie, 2002;

Bengtson and Roth, 2008). This family of approaches, known as "mention-pair" models, has several drawbacks, including that the number of predictions to be made scales quadratically in the number of mentions, and, more fundamentally, that mention-pair models are extremely coarse models of coreference. For instance, a mention-pair model does not distinguish between the labeling of a pair of coreferent mentions that are close together in the document and that of a pair of coreferent mentions that are hundreds of words apart; for the purposes of a mention pair model, each pair has the same label (namely, 1) so long as the pairs are part of the same cluster in y.

MENTION-RANKING MODELS Mention-ranking models (Denis and Baldridge, 2008; Rahman and Ng, 2009; Durrett and Klein, 2013; Chang et al., 2013) are an alternative unstructured approach to coreference resolution, which attempt to address some of the shortcomings of mention-pair models. In particular, mention-ranking models attempt to predict either a single, coreferent *antecedent* mention x_j for each mention x_n in the document, or that x_n has no coreferent antecedent mentions. Formally, mention ranking models predict $\hat{a}_n \in = \{1, \ldots, n-1, \epsilon\}$ for each mention x_n. Here, $\hat{a}_n$ is either the index of a mention x_j such that $j < n$, or it is the symbol ϵ, which denotes the dummy antecedent. Predicting a dummy antecedent allows the model to capture the fact that a mention x_n may not be coreferent with any preceding mention. After predicting antecedents $\{\hat{a}_n\}_{n=1}^{N}$, a full clustering $\hat{y}$ can be produced by taking the weakly connected components of the graph consisting of nodes x_n and directed edges $\{(x_n, x_{\hat{a}_n}) \mid \hat{a}_n \neq \epsilon\}$.

STRUCTURED MODELS Historically, structured prediction approaches to coreference resolution have either attempted to predict the coreference clustering itself (McCallum and Wellner, 2003; Culotta et al., 2007; Poon and Domingos, 2008; Haghighi and Klein, 2010; Stoyanov and Eisner, 2012; Cai and Strube, 2010), or to predict an auxiliary structure from which the clusters can be extracted. For instance, much recent work (Yu and Joachims, 2009; Fernandes et al., 2012; Björkelund and Kuhn, 2014; Martschat and Strube, 2015) has focused on predicting a directed "coreference tree"; clusters can then be extracted by taking the tree's weakly connected components, as in mention-ranking.

3.2 A Neural Mention-Ranking Model

In motivating our baseline mention-ranking model, we note that one of the major challenges that has been associated with resolving coreference is that in typical documents the number of mentions that are *non-anaphoric* – that is, that are not coreferent with any previous mention – far exceeds the number of mentions that are anaphoric (Kummerfeld and Klein, 2013; Durrett and Klein, 2013). This preponderance of non-anaphoric mentions makes coreference resolution challenging, partly because many basic coreference features, such as those looking at head, number, or gender match fail to distinguish between truly coreferent pairs and the large number of matching but nonetheless non-coreferent pairs. Indeed, several authors have noted that it is difficult to obtain good performance on the coreference task using simple features (Lee et al., 2011; Fernandes et al., 2012; Durrett and Klein, 2013; Kummerfeld and Klein, 2013; Björkelund and Kuhn, 2014) and, as a result, state-of-the-art systems tended to use linear models with complicated feature conjunction schemes in order to capture more fine-grained interactions. While this approach has shown success, it is not obvious which additional feature conjunctions will lead to improved performance, which is problematic as systems attempt to scale with new data and features.

In this section, we propose a data-driven model for coreference that does not require pre-specifying any feature relationships. Inspired by recent work in learning representations for natural language tasks (Collobert et al., 2011), we explore neural network models which take only raw, unconjoined features as input, and attempt to learn intermediate representations automatically. In particular, the model we describe attempts to create independent feature representations useful for both detecting the anaphoricity of a mention (that is, whether or not a mention is anaphoric) and ranking the potential antecedents of an anaphoric mention. Adequately capturing anaphoricity information has long been thought to be an important aspect of the coreference task (see Ng (2004)), since a strong non-anaphoric signal might, for instance, discourage the erroneous prediction of an antecedent for a non-anaphoric mention even in the presence of a misleading head match.

We furthermore attempt to encourage the learning of the desired feature representations by pre-training the model's weights on two corresponding subtasks, namely, anaphoricity detection and antecedent ranking of known anaphoric mentions. On its publication, the model presented in this section had an absolute gain of almost 2 points in CoNLL score over a similar but linear mention-ranking model on the CoNLL 2012 English test set (Pradhan et al., 2012), and of over 1.5 points over the state-of-the-art coreference system at the time.

3.2.1 Mention Ranking Details

As we noted in the introduction to this chapter, mention-ranking systems aim to identify whether a mention is coreferent with an antecedent mention, or whether it is instead non-anaphoric (the first mention in the document referring to a particular entity). This is accomplished by assigning a score to the mention's potential antecedents as well as to the possibility that it is non-anaphoric, and then predicting the greatest scoring option. Letting $\mathcal{M}$ refer to the set of possible document mentions, a mention-ranking model defines a scoring function $f(x_n, a) : \mathcal{M} \times \{1, \ldots, N, \epsilon\} \rightarrow \mathbb{R}$, and predicts the antecedent of x_n to be

$$\hat{a}_n = \operatorname*{arg\,max}_{a \in \{1, \ldots, n-1, \epsilon\}} f(x_n, a).$$

It is common to be quite liberal when extracting the set of mentions $\mathcal{M}(x)$, taking, essentially, every noun phrase or pronoun to be a candidate mention, so as not to prematurely discard those that might be coreferent (Lee et al., 2011; Fernandes et al., 2012; Chang et al., 2012; Durrett and Klein, 2013). For instance, the Berkeley Coreference System (herein BCS) (Durrett and Klein, 2013), which we use for mention extraction in our experiments, recovers approximately 96.4% of the truly anaphoric mentions in the CoNLL 2012 training set, with an almost 3.5:1 ratio of non-anaphoric mentions to anaphoric mentions among the extracted mentions.

The structural simplicity of the mention-ranking framework puts much of the burden on the scoring function $f(x_n, a)$. We begin by considering mention-ranking systems using linear scoring functions. We will then extend these models to operate over learned non-linear representations.

Linear mention-ranking models generally utilize the following scoring function

$$f_{\text{lin}}(x_n, a) = \boldsymbol{w}^{\mathsf{T}} \boldsymbol{\phi}(x_n, a),$$

where $\boldsymbol{\phi} : \mathcal{M} \times \{1, \ldots, N, \epsilon\} \rightarrow \mathbb{R}^d$ is a pairwise feature function defined on a mention and a potential antecedent, and $\boldsymbol{w}$ is a learned parameter vector.

To add additional flexibility to the model, linear mention ranking models may duplicate individual features in $\boldsymbol{\phi}$, with one version being used when predicting an antecedent for x_n, and another when predicting that x_n is non-anaphoric (Durrett and Klein, 2013). Such a scheme effectively gives rise to the following piecewise scoring function

$$f_{\text{lin+}}(x_n, a) \triangleq \begin{cases} \boldsymbol{u}^{\mathsf{T}} \left[\begin{smallmatrix} \boldsymbol{\phi}_{\text{a}}(x_n) \\ \boldsymbol{\phi}_{\text{p}}(x_n, a) \end{smallmatrix} \right] & \text{if } a \neq \epsilon \\ \\ \boldsymbol{v}^{\mathsf{T}} \boldsymbol{\phi}_{\text{a}}(x_n) & \text{if } a = \epsilon, \end{cases} \tag{3.1}$$

where $\boldsymbol{\phi}_{\text{a}} : \mathcal{M} \rightarrow \mathbb{R}^{d_{\text{a}}}$ is a feature function defined on a mention and its context, $\boldsymbol{\phi}_{\text{p}} : \mathcal{M} \times \{1, \ldots, N\} \rightarrow \mathbb{R}^{d_{\text{p}}}$ is a pairwise feature function defined on a mention and a potential antecedent, and parameters $\boldsymbol{u}$ and $\boldsymbol{v}$ replace $\boldsymbol{w}$. Above, we have made an explicit distinction between pairwise features ($\boldsymbol{\phi}_{\text{p}}$) and those strictly on x and its context ($\boldsymbol{\phi}_{\text{a}}$), and moreover assumed that our features need not examine potential antecedents when predicting $a = \epsilon$.

We refer to the basic, unconjoined features used for $\boldsymbol{\phi}_{\text{a}}$ and $\boldsymbol{\phi}_{\text{p}}$ as *raw* features. Figure 3.2 shows two versions of these features, a base set Basic and an extended set Basic+. The Basic set are the raw features used in BCS, and Basic+ includes additional raw features used in other recent coreference sytems. For instance, Basic+ additionally includes features suggested by Recasens et al. (2013) to be useful for anaphoricity, such as the number of a mention, its named entity status, and its animacy, as well as number and gender information. We additionally include bilexical head features, which are used in many well-performing systems (for instance, that of Fernandes et al. (2012)).

Mention Features (ϕ_{a})

	Feature	Value Set
	Mention Head	$\mathcal{V}$
	Mention First Word	$\mathcal{V}$
	Mention Last Word	$\mathcal{V}$
	Word Preceding Mention	$\mathcal{V}$
	Word Following Mention	$\mathcal{V}$
	# Words in Mention	$\{1, 2, \ldots\}$
	Mention Synt. Ancestry	see BCS (2013)
	Mention Type	$\mathcal{T}$
+	Mention Governor	$\mathcal{V}$
+	Mention Sentence Index	$\{1, 2, \ldots\}$
+	Mention Entity Type	NER tags
+	Mention Number	{sing.,plur.,unk}
+	Mention Animacy	{an.,inan.,unk}
+	Mention Gender	{m,f,neut.,unk}
+	Mention Person	{1,2,3,unk}

Pairwise Features (ϕ_{p})

	Feature	Value Set
	Basic features on Mention	see above
	Basic features on Antecedent	see above
	Mentions between Ment., Ante.	{0...10}
	Sentences between Ment., Ante.	{0...10}
	i-within-i	{T,F}
	Same Speaker	{T,F}
	Document Type	{Conv.,Art.}
	Ante., Ment. String Match	{T,F}
	Ante. contains Ment.	{T,F}
	Ment. contains Ante.	{T,F}
	Ante. contains Ment. Head	{T,F}
	Mention contains Ante. Head	{T,F}
	Ante., Ment. Head Match	{T,F}
	Ante., Ment. Synt. Ancestries	see above
+	Basic+ features on Ment.	see above
+	Basic+ features on Ante.	see above
+	Ante., Ment. Numbers	see above
+	Ante., Ment. Genders	see above
+	Ante., Ment. Persons	see above
+	Ante., Ment., Entity Types	see above
+	Ante., Ment. Heads	see above
+	Ante., Ment. Types	see above

Figure 3.2: Features used for $\phi_{\mathrm{a}}(x_n)$ and $\phi_{\mathrm{p}}(x_n, a)$. The '+' indicates a feature is in Basic+ feature set. $\mathcal{V}$ denotes the training vocabulary, and $\mathcal{T}$ denotes the set of mention types, viz., {nominal,proper} ∪ {canonical pronouns}, as defined in BCS. Conv. and Art. abbreviate conversation and article (resp.). Lexicalized features occurring fewer than 20 times in the training set back off to part-of-speech; bilexical heads occurring fewer than 10 times back off to an indicator feature. Animacy information is taken from a list and rules used in the Stanford Coreference system (Lee et al., 2013).

Problems with Raw Features

Many authors have observed that, taken individually, raw features tend to not be particularly predictive for the coreference task. We examine this phenomenon empirically in Figure 3.3.

These graphs show that the vast majority of individual features do not give a strong positive signal either of anaphoricity or for an antecedent match. To address this issue, state-of-the-art mention-ranking systems often rely on manual or otherwise induced conjunction schemes to capture specific feature interactions. Durrett and Klein (2013), for instance, conjoin all raw features in ϕ_{a} with the *type* of the mention x, and all raw features in ϕ_{p} with the types of the current mention and antecedent. For these purposes, the *type* of a mention is either "nominal", "proper", or a canonicalization of the pronoun if it is a pronominal mention. Fernandes et al. (2012) and Björkelund and Kuhn (2014) use an automatic but complicated scheme to induce conjunctions by first extracting feature templates from a separately trained decision tree, and then doing greedy forward selection among the templates. These conjunctions add some non-linearity to the scoring function while still maintaining a tractable, though large, feature set.

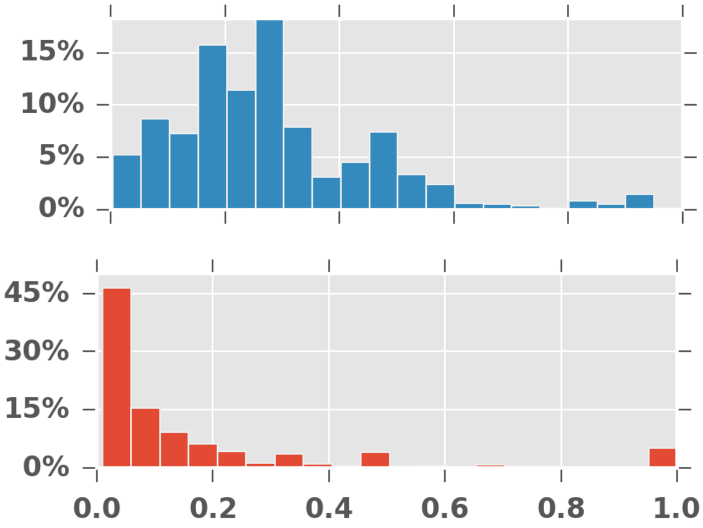


Figure 3.3: Two histograms illustrating the predictive ability of raw (unconjoined) features per feature occurrence: (top) mention-context features from ϕ_{a} as independent predictors of anaphoricity ($a \neq \epsilon$), and (bottom) antecedent-mention features from ϕ_{p} as independent predictors of coreferent mentions. Very few raw features are strong indicators of either anaphoricity or an antecedent match. Data taken from the CoNLL development set.

3.2.2 Learning Features for Ranking

As an alternative to the aforementioned feature conjunction schemes, we consider learning feature representations in order to better capture relevant aspects of the task. Representation learning affords the model more flexibility in exploiting feature interactions, although it can make the underlying training problem more difficult.

Model

We use a neural network to define our model as an extension to the mention-ranking model introduced in Section 3.2.1. We consider in particular the scoring function:

$$f(x_n, a) \triangleq \begin{cases} \boldsymbol{u}^\top \boldsymbol{g}(\left[\begin{smallmatrix} \boldsymbol{h}_{\mathrm{a}}(x_n) \\ \boldsymbol{h}_{\mathrm{p}}(x_n, a) \end{smallmatrix}\right]) + u_0 & \text{if } a \neq \epsilon \\ \\ \boldsymbol{v}^\top \boldsymbol{h}_{\mathrm{a}}(x_n) + v_0 & \text{if } a = \epsilon, \end{cases} \tag{3.2}$$

where $\boldsymbol{h}_{\mathrm{a}}$ and $\boldsymbol{h}_{\mathrm{p}}$ are feature representations, non-linear functions of the features $\boldsymbol{\phi}_{\mathrm{a}}$ and $\boldsymbol{\phi}_{\mathrm{p}}$ (respectively), and $\boldsymbol{g}$ is a function of these representations. In particular, we define

$$\boldsymbol{h}_{\mathrm{a}}(x_n) = \tanh(\boldsymbol{W}_{\mathrm{a}}\, \boldsymbol{\phi}_{\mathrm{a}}(x_n) + \boldsymbol{b}_{\mathrm{a}}) \tag{3.3}$$

$$\boldsymbol{h}_{\mathrm{p}}(x_n, a) = \tanh(\boldsymbol{W}_{\mathrm{p}}\, \boldsymbol{\phi}_{\mathrm{p}}(x_n, a) + \boldsymbol{b}_{\mathrm{p}}), \tag{3.4}$$

and we take $\boldsymbol{g}$ to either be the identity function, in which case the above model is analogous to $s_{\mathrm{lin+}}$ as defined in (3.1) but defined over non-linear feature representations, or to be an additional hidden layer:

$$\boldsymbol{g}(\left[\begin{smallmatrix}\boldsymbol{h}_{\mathrm{a}}(x_n)\\ \boldsymbol{h}_{\mathrm{p}}(x_n,a)\end{smallmatrix}\right]) = \tanh(\boldsymbol{W}\left[\begin{smallmatrix}\boldsymbol{h}_{\mathrm{a}}(x_n)\\ \boldsymbol{h}_{\mathrm{p}}(x_n,a)\end{smallmatrix}\right] + \boldsymbol{b}).$$

For ease of exposition, we will refer to these two settings of $\boldsymbol{g}$ as $\boldsymbol{g}_1$ and $\boldsymbol{g}_2$ (respectively) in what follows. As we will see below, both settings lead to comparable performance, but to a different error distribution.

In either case, by defining the functions $\boldsymbol{h}_{\mathrm{a}}$ and $\boldsymbol{h}_{\mathrm{p}}$, we allow the model to learn representations of the input features $\boldsymbol{\phi}_{\mathrm{a}}$ and $\boldsymbol{\phi}_{\mathrm{p}}$. The benefit of the added non-linearities is that, in theory, it is no longer necessary to explicitly specify feature conjunctions, since the model may learn them automatically if necessary. Accordingly, for this model we use only $\boldsymbol{\phi}_{\mathrm{a}}$ and $\boldsymbol{\phi}_{\mathrm{p}}$ consisting of the raw features in Figure 3.2 without conjunctions. Any interaction between these features must be learned by the feature representations $\boldsymbol{h}_{\mathrm{p}}$ and $\boldsymbol{h}_{\mathrm{a}}$.

3.2.3 Training

We can directly train our model using back-propagation. To specify the training objective, we first define

$$\mathcal{C}(x_n) = \begin{cases} \{j < n \mid x_j \text{ and } x_n \text{ are coreferent}\} & \text{if } x_n \text{ is anaphoric} \\ \\ \{\epsilon\} & \text{otherwise.} \end{cases}$$

Additionally, we define

$$y_n^{\ell} = \operatorname*{arg\,max}_{a \in \mathcal{C}(x_n)} f(x_n, a) \tag{3.5}$$

That is, y_n^ℓ is the highest scoring correct antecedent of x_n, which may be ϵ. Thus, following recent work (Yu and Joachims, 2009; Fernandes et al., 2012; Chang et al., 2013; Durrett and Klein, 2013), we view each mention as having a "latent antecedent," which renders the objectives of even models with a linear scoring function non-convex. We train to minimize the regularized, slack-rescaled, latent-variable loss given by:

$$L(\theta) = \sum_{n=1}^{N} \max_{a \in \{1,\ldots,n-1,\epsilon\}} \Delta(x_n, a)(1 + s(x_n, a) - s(x_n, y_n^\ell)) + \lambda ||\theta||_1, \tag{3.6}$$

where Δ is a mistake-specific cost function, which is 0 when $a \in \mathcal{C}(x_n)$. Above, we use θ to refer to the full set of parameters $\{\boldsymbol{W}, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{W}_{\text{a}}, \boldsymbol{W}_{\text{p}}, \boldsymbol{b}_{\text{a}}, \boldsymbol{b}_{\text{p}}\}$.

For experiments, we define Δ to take on different costs for the three kinds of mistakes possible in a coreference task, as follows:

$$\Delta(x, a) = \begin{cases} \alpha_1 & \text{if } a \neq \epsilon \wedge \epsilon \in \mathcal{C}(x) \\ \alpha_2 & \text{if } a = \epsilon \wedge \epsilon \notin \mathcal{C}(x) \\ \alpha_3 & \text{if } a \neq \epsilon \wedge a \notin \mathcal{C}(x). \end{cases} \tag{3.7}$$

The α_i determine the trade-off between these mistakes (and thus precision and recall). Adopting the terminology of BCS, we refer to these mistakes as "false link" (FL), "false new" (FN), and "wrong link" (WL), respectively.

3.2.4 Representations from Subtasks

While we could train our full model directly, it is known to be difficult to train high performing non-convex neural-network models from a random initialization (Erhan et al., 2010). In order to overcome the problems associated with training from this setting, and to learn feature representations useful for the full coreference task, we pretrain subparts of the model on the subtasks targeting the desired feature representations. We then train the entire model on the full coreference task (from the pre-trained initializations). As we will see, the pre-training scheme outlined below helps the model achieve improved performance.

The proposed pre-training scheme involves learning the parameters associated with $\boldsymbol{h}_{\mathrm{a}}$ and $\boldsymbol{h}_{\mathrm{p}}$ using two natural subtasks: anaphoricity detection and antecedent ranking. In particular, we (1) train $\boldsymbol{h}_{\mathrm{a}}$ on the task of predicting whether a particular mention is anaphoric or not, and (2) train $\boldsymbol{h}_{\mathrm{p}}$ on the task of predicting the antecedent of mentions known to be anaphoric.

Anaphoricity Detection

For the first subtask we attempt to predict whether a mention is anaphoric or not based only on its local context.[3] Anaphoricity detection in various forms has been used as an initial step in several coreference systems (Ng and Cardie, 2002; Bengtson and Roth, 2008; Rahman and Ng, 2009; Björkelund and Farkas, 2012), and the related question of whether a mention can be determined to be a *singleton* or not has been explored recently by Recasens et al. (2013), Ma et al. (2014), and others. Note, however, that singleton detection is slightly different from anaphoricity detection, since a mention can be non-anaphoric but not a singleton if it is the first mention in a cluster.

Formally, let $t_n \in \{-1, 1\}$ indicate whether $\epsilon \in \mathcal{C}(x_n)$ or not (respectively). That is, $t_n = 1$

[3] While performance on this *subtask* can in fact be improved further by looking at previous mentions, features learned in this way led to inferior performance on the full task.

Feat. (Conj.)	Model	Anaphoric			Ante
		P	R	F_1	Acc.
Basic (N)	Lin.	74.15	74.20	74.18	69.10
Basic (Y)	Lin.	73.98	75.04	74.51	79.76
Basic (N)	NN	75.30	75.36	75.33	81.65
Basic+ (N)	Lin.	74.14	74.71	74.43	74.02
Basic+ (Y)	Lin.	74.24	75.39	74.81	80.44
Basic+ (N)	NN	75.84	76.02	75.93	82.86

Table 3.1: Performance of the two subtasks on the CoNLL 2012 development set by feature set and model type. ``Conj." indicates whether conjunctions are used. The linear anaphoric system is an SVM (LibLinear implementation (Fan et al., 2008)), and the linear antecedent system is a linear model with the margin-based objective.

if and only if x_n is anaphoric. Define the subtask scoring function $f_a : \mathcal{M} \rightarrow \mathbb{R}$ as

$$f_{\mathrm{a}}(x_n) = \boldsymbol{v}_{\mathrm{a}}{}^{\mathsf{T}} \boldsymbol{h}_{\mathrm{a}}(x_n) + \nu_0,$$

where the vector $\boldsymbol{v}_{\mathrm{a}}$ and the bias ν_0 are specific to this subtask and are discarded after pretraining.

We train this model to minimize the following slack-rescaled objective

$$L_{\mathrm{a}}(\theta_{\mathrm{a}}) = \sum_{n=1}^{N} \Delta_{\mathrm{a}}(t_n)[1 - t_n\, f_{\mathrm{a}}(x_n)]_+ + \lambda ||\theta_{\mathrm{a}}||_1,$$

where Δ_{a} is a class-specific cost used to help encourage anaphoric decisions given the imbalanced data set, and $\theta_{\mathrm{a}} = \{\boldsymbol{v}_{\mathrm{a}}, \boldsymbol{W}_{\mathrm{a}}, \boldsymbol{b}_{\mathrm{a}}\}$ are the parameters of the subtask.

Antecedent Ranking

For the second subtask, antecedent ranking, we predict the antecedent for mentions known a priori to be anaphoric. This subtask is inspired by the "gold mention" version of the coreference task. Systems designed for this task are forced to handle many fewer non-anaphoric mentions and can often successfully utilize richer feature representations. The setup for this task is similar to the full coreference problem, except that we discard any mention x_n such that $\epsilon \in \mathcal{C}(x_n)$. Thus, we define the pairwise scoring function $f_\mathrm{p} : \mathcal{M} \times \{1, \ldots, N\} \rightarrow \mathbb{R}$ as

$$f_\mathrm{p}(x_n, a) = \boldsymbol{u}_\mathrm{p}{}^\mathsf{T} \boldsymbol{h}_\mathrm{p}(x_n, a) + \upsilon_0.$$

As before, $\boldsymbol{u}_\mathrm{p}$ and υ_0 are discarded after training for this subtask, but we keep the rest of the parameters. For training, we use an analogous latent-variable loss function to that used for the full coreference task, except that a may not be ϵ, and the cost $\Delta(x, a)$ is always 1 (when it is nonzero).

Subtask Performance

As a preliminary experiment, we train models for these two subtasks using both the Basic and Basic+ raw features. Table 3.1 shows the results. For the first subtask, experiments look at the precision, recall, and F_1 score of predicting anaphoric mentions on the CoNLL 2012 development set. As a baseline we use an L1-regularized SVM implemented using LibLinear (Fan et al., 2008), both using raw features and using features conjoined according to the BCS scheme. For the second subtask, experiments look at the accuracy of the model in predicting the correct antecedent on known anaphoric mentions. As a baseline we use a linear mention ranking model, with and without conjunctions, trained using the same margin-based loss.

In both subtasks, the neural network model performs quite well, significantly better than the unconjoined baselines and better than the model trained with manually conjoined features. We provide a visual representation of the antecedent ranking features learned in Figure 3.4.

While the improved subtask performance does not imply better performance on the full coreference task, it shows that model can learn useful feature representations with only raw input features.

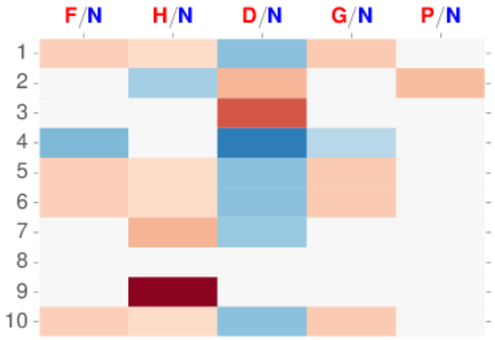


Figure 3.4: Visualization of the representation matrix $\boldsymbol{W}_{\mathrm{p}}$. A subset of the raw features were manually grouped into five classes indicating: full lexical match [F], head match [H], mention/sentence distance [D] (near versus far), gender/number match [G], and type [P] (pronoun versus other). The heat map illustrates 10-columns of $\boldsymbol{W}_{\mathrm{p}}$ as a weighted combination of these classes, roughly illustrating the combination of raw features required for this dimension of the representation.

3.2.5 Main Experiments

Our experiments examine performance as compared with other coreference systems, as well as the effect of features, pre-training, and model architecture. We also perform a qualitative comparison of our model with the analogous linear model on some challenging non-anaphoric cases.

Methods

All experiments use the CoNLL 2012 English dataset (Pradhan et al., 2012), which is based on the OntoNotes corpus (Hovy et al., 2006). The data set contains 3,493 documents consisting of 1.6 million words. We use the standard experimental split with the training set containing 2,802 documents and 156K annotated mentions, the development set containing 343 documents and 19K annotated mentions, and the test set containing 348 documents and 20K annotated mentions. For all experiments, we use BCS Durrett and Klein (2013) to extract system mentions and to compute some of the features.

For training, we minimize the loss described above using the composite mirror descent AdaGrad update (Duchi et al., 2011) with document sized mini-batches. We tuned the AdaGrad learning rate and regularization parameters using a grid search over possible learning rates $\eta \in \{0.001, 0.002, 0.01, 0.02, 0.1, 0.2\}$ and over regularization parameters $\lambda \in \{10^{-6}, \dots, 10^{-1}\}$. For the full coreference task, we use a different learning rate for the pre-trained weights and for the second-layer weights, using $\eta_1 = 0.1$ and $\eta_2 = 0.001$, respectively, and $\lambda = 10^{-6}$. When initializing weight-matrices that were not pre-trained we used the sparse initialization technique proposed by Sutskever et al. (2013). For all experiments we use the cost-weights $\boldsymbol{\alpha} = \langle 0.5, 1.2, 1 \rangle$ in defining Δ.

For the anaphoricity representations the matrix dimensions used are $\boldsymbol{W}_{\mathrm{a}} \in \mathbb{R}^{128 \times d_{\mathrm{a}}}$, and for the pairwise representations the matrix dimensions used are $\boldsymbol{W}_{\mathrm{p}} \in \mathbb{R}^{700 \times d_{\mathrm{p}}}$. In the $\boldsymbol{g}_2$ model, the outer matrix dimensions are $\boldsymbol{W} \in \mathbb{R}^{128 \times (d_{\mathrm{p}} + d_{\mathrm{a}})}$. With the Basic+ features, d_{p} and d_{a} come out to be slightly less than 10^6 and 10^4, respectively, with bilexical head features accounting for the vast majority of d_{p}. Note that the BCS conjunction scheme, for instance, applied to our raw features gives a d_{p} and d_{a} that are over an order of magnitude larger.[4] We tuned all hyper-parameters (as well as those of baseline systems) on the development set.

We use the CoNLL 2012 scoring script (Pradhan et al., 2014; Luo et al., 2014) v8.01 (`http://conll.github.io/\reference-coreference-scorers/`), which scores based on 3 metrics, including MUC (Vilain et al., 1995), CEAF$_e$ (Luo, 2005), and B^3 (Bagga and Baldwin, 1998),

[4] The next section shows that it is possible to do at least as well under this model without any bilexical head features at all, and the feasibility of discarding features altogether was preliminarily explored in Wiseman et al. (2016a).

as well as the CoNLL score, which is the arithmetic mean of the 3 metrics. Code implementing our models is available at `https://github.com/swiseman/nn_coref`. The system trains in time comparable to that of linear systems, mainly because we use only raw features and sparse margin-based gradient updates.

Results

System	MUC			B^3			CEAF$_e$			
	P	R	F$_1$	P	R	F$_1$	P	R	F$_1$	CoNLL
BCS	74.89	67.17	70.82	64.26	53.09	58.14	58.12	52.67	55.27	61.41
Ma	81.03	66.16	**72.84**	66.9	51.10	57.94	68.75	44.34	53.91	61.56
B&K	74.3	67.46	70.72	62.71	54.96	58.58	59.4	52.27	55.61	61.63
D&K	72.73	69.98	71.33	61.18	56.60	58.80	56.20	54.31	55.24	61.79
(3.2), $\boldsymbol{g}_2$	76.96	68.10	72.26	66.90	54.12	59.84	59.02	53.34	56.03	62.71
(3.2), $\boldsymbol{g}_1$	76.23	69.31	72.60	66.07	55.83	**60.52**	59.41	54.88	**57.05**	**63.39**

Table 3.2: Results on CoNLL 2012 English test set. We compare against recent state-of-the-art systems, including (in order) Durrett and Klein (2013), Ma et al. (2014), Björkelund and Kuhn (2014), and Durrett and Klein (2014) (rescored with the v8.01 scorer). F$_1$ gains are significant ($p < 0.05$ under the bootstrap resample test (Koehn, 2004)) compared with both B&K and D&K for all metrics.

Our main results are shown in Table 3.2. This table compares the performance of our system with the performance reported by several other state-of-the art systems on the CoNLL 2012 English coreference test set. Our full models achieved the best F$_1$ score across two of the three metrics and have the best aggregate (CoNLL) score, with an improvement of over 1.5 points over the best reported result, and of almost 2 points over the best mention-ranking system. Our F$_1$ improvements on all three metrics are significant ($p < 0.05$ under the bootstrap resample test (Koehn, 2004)) as compared with both Björkelund and Kuhn (2014), and Durrett and Klein (2014), the two most recent, state-of-the-art systems.

Since our full models use some additional raw features (although an order of magnitude fewer total features than the comparable conjunction-based linear model), we are interested

Model	Features	MUC	B^3	CEAF$_e$	CoNLL
Lin.		70.44	59.10	55.57	61.71
NN ($\boldsymbol{g}_2$)	Basic	71.59	60.56	57.45	63.20
NN ($\boldsymbol{g}_1$)		71.86	60.9	57.90	63.55
Lin.		70.92	60.05	56.39	62.45
NN ($\boldsymbol{g}_2$)	Basic+	72.68	61.70	58.32	64.23
NN ($\boldsymbol{g}_1$)		72.74	61.77	58.63	64.38

Table 3.3: F_1 performance comparison between state-of-the-art linear mention-ranking model (Durrett and Klein, 2013) and our full models on CoNLL 2012 development set for different feature sets.

in what part of the improvement in performance comes from features rather than modeling power. Table 3.3 compares the full model to BCS, a system effectively using the $s_{\text{lin+}}$ scoring function together with a manual conjunction scheme, on both Basic and Basic+ features. While our models outperform BCS in both cases, we see that as we add more features (as in the Basic+ set), the performance gap between our model and the linear system becomes even more pronounced.

We may also wonder whether the architecture represented by our scoring function, where the intermediate representations $\boldsymbol{h}_{\text{a}}$ and $\boldsymbol{h}_{\text{p}}$ are separated in the first layer, is necessary for these results. We accordingly compare with the fully connected versions of these two models (which are equivalent to 1 and 2 layer multi-layer perceptrons) using the Basic+ features in Table 3.4.[5] There, we also evaluate the effect of pre-training on these models by comparing with the results of training from a random initialization. We see that while even randomly initialized models are capable of excellent performance, pre-training is beneficial, especially for $\boldsymbol{g}_1$.

[5] We also experimented with bilinear models both with and without non-linearities; these were also inferior.

Model	MUC	B^3	$CEAF_e$	CoNLL
Fully Conn. 1 Layer	71.80	60.93	57.51	63.41
Fully Conn. 2 Layer	71.77	60.84	57.05	63.22
$\boldsymbol{g}_1$ + RI	71.92	61.06	57.59	63.52
$\boldsymbol{g}_1$ + PT	72.74	61.77	58.63	64.38
$\boldsymbol{g}_2$ + RI	72.31	61.79	58.06	64.05
$\boldsymbol{g}_2$ + PT	72.68	61.70	58.32	64.23

Table 3.4: Comparison of performance (in F_1 score) of various models on CoNLL 2012 development set using Basic+ features. ``PT'' and ``RI'' refer to pretraining and random initialization respectively. ``Fully Conn.'' refers to baseline fully connected networks. See text for further model descriptions.

3.2.6 Discussion

We attempt to gain insight into our model's errors using using two different error breakdowns. In Table 3.5 we show the errors as reported by the analysis tool of Kummerfeld and Klein (2013). In Table 3.6 we show a more fine-grained breakdown inspired by a similar analysis in Durrett and Klein (2013). In the latter table, we categorize the errors made by our system on the CoNLL 2012 development data in terms of (1) whether or not the mention has a head match with a previously occurring mention in the document, unless it is a pronominal mention, which we treat separately, (2) in terms of the status of the mention in the gold clustering, namely, singleton, first-in-cluster, or anaphoric, and (3) in terms of the type of error made (which, as discussed in Section 3.2.2, are one of FL, FN, and WL).

We note that the two models have slightly different error profiles, with $\boldsymbol{g}_1$ being slightly better at recall and $\boldsymbol{g}_2$ being slightly better at precision. Indeed, we see from Table 3.6 that the two models make a comparable number of total errors ($\boldsymbol{g}_1$ makes only 17 fewer errors overall). The increased precision of the $\boldsymbol{g}_2$ model is presumably due to the second layer around $\boldsymbol{h}_{\mathrm{a}}$ and $\boldsymbol{h}_{\mathrm{p}}$ in $\boldsymbol{g}_2$ allowing for antecedent evidence to interact with anaphoricity evidence in a more complicated way. Ultimately, however, coreference systems operating over system men-

Error Type	BCS	NN ($\boldsymbol{g}_1$)	NN ($\boldsymbol{g}_2$)
Conflated Entities	1603	1434	1371
Extra Mention	651	568	529
Extra Entity	655	623	561
Divided Entity	1989	1837	1835
Missing Mention	1004	997	1005
Missing Entity	1070	1026	1114

Table 3.5: Absolute error counts from the coreference analysis tool of Kummerfeld and Klein (2013). The upper set roughly corresponds to the precision and the lower to the recall of the coreference clusters produced by the model.

NN ($\boldsymbol{g}_1$)	Singleton		1^{st} in clust.		Anaphoric	
	FL	#	FL	#	FN + WL	#
HM	817	8.2K	147	0.8K	700 + 318	4.7K
No HM	86	19.8K	41	2.4K	677 + 59	1.0K
Pron.	948	2.6K	257	0.5K	434 + 875	7.3K
NN ($\boldsymbol{g}_2$)	**Singleton**		**1^{st} in clust.**		**Anaphoric**	
	FL	#	FL	#	FN + WL	#
HM	770	8.2K	130	0.8K	803 + 306	4.7K
No HM	73	19.8K	39	2.4K	699 + 52	1.0K
Pron.	896	2.6K	249	0.5K	456 + 869	7.3K

Table 3.6: Errors made by NN ($\boldsymbol{g}_1$) (top) and NN ($\boldsymbol{g}_2$) (bottom) on CoNLL 2012 English development data. Rows correspond to (1) mentions with a (previous) head match (hm), that is, mentions x such that $\mathcal{A}(x)$ contains another mention with the same head word, (2) with no previous head match (no hm), and (3) to pronominal mentions, respectively. The 3 column groups correspond to singleton, first-in-cluster, and anaphoric mentions (resp.), as determined by the gold clustering, with the number and type of errors on the left and the total number of mentions in the category (#) on the right.

tions are already biased toward precision, and so the increased precision of $\boldsymbol{g}_2$ is not as helpful as the increased recall of $\boldsymbol{g}_1$ in the final CoNLL score.

In further analysis we found that many of the correct predictions made by the $\boldsymbol{g}_2$ model not made by $\boldsymbol{g}_1$ and the linear model involve predicting non-anaphoric even in the presence of highly misleading antecedent features like head-match. Figure 3.5 shows some examples of mentions with previous head matches that the linear system predicted as anaphoric and that our system correctly identifies as non-anaphoric.

Non-Anaphoric (x)	Spurious Antecedent (y)
the Nika TV company	an independent company
Lexus sales	GM 's domestic car sales
The storage area	the harbor area
the Budapest location	Radio Free Europe 's new location
the synagogue	the synagogue too or something
the equity market	The junk market
their silver coin	one silver coin
the international school	The Hong Kong elementary school
the 1970s	the early 1970s
the 2003 season	the 2001 season

Figure 3.5: Example mentions x that were correctly marked non-anaphoric by $\boldsymbol{g}_2$, but incorrectly marked anaphoric with y as an antecedent by the Basic+ linear model. These examples highlight the difficult case where there is a spurious head-match between non-coreferent pairs. See text for further details.

We illustrate how the features in Figure 3.2 might be useful in such cases by considering the first example in Figure 3.5. There, a comma follows “the Nika TV company” in the text (and is picked up by the “word following” feature), perhaps indicating an appositive, which makes anaphoricity unlikely. The model can also learn that the “company-company” head match is often misleading, and, in general, distance features may also rule out head matches. Note that while these features on their own may be more or less correlated with a mention being non-anaphoric, the model learns to combine them in a predictive way.

Further Improving Coreference Systems

Table 3.6 also gives a sense of where coreference systems such as ours need to improve. It is first important to note that the case of resolving an anaphoric mention that has no previous head matches (e.g., identifying that “the team” and “the New York Giants” are coreferent), which is often taken to be one of the major challenges facing coreference systems because it presumably requires semantic information, is not the largest source of errors. In fact, we see from Table 3.6 (second row, third column in both sub-tables) that while these cases do indeed account for a substantial percentage of errors, we make hundreds more errors predicting singleton pronominal mentions to be anaphoric (in the case of $\boldsymbol{g}_1$) and on incorrectly linking anaphoric pronominal mentions (in the case of $\boldsymbol{g}_2$). Further analysis indicates that these errors are almost entirely related to incorrectly linking pleonastic pronouns, such as “it” or “you,” and that moreover the incorrectly predicted antecedent for these pleonastic pronouns is almost always (another instance of) the same pronoun.

That these pleonastic cases are so problematic is interesting when considered against the backdrop of the prediction strategies typically employed by coreference systems. In particular, one might suspect, that structured approaches to coreference resolution, which do not predict an antecedent for each mention independently, might allow us to capture the fact that, for instance, a cluster of coreferent mentions should generally not consist solely of pronouns, and thereby avoid predicting (identical) pronominal antecedents for pleonastic pronouns. Such an approach is explored in the next section.

3.3 A Neural, Structured Coreference Model

We noted at the end of the previous section that structured, non-local coreference models would seem to hold promise for avoiding many common coreference errors. At the same time, the results of employing such models in practice have been decidedly mixed, and, through 2015, state-of-the-art results had been obtained using completely local, mention-ranking systems, as in the previous section. We argue that informative *cluster*, rather than mention, level features are very difficult to devise, limiting their effectiveness. Accordingly, we instead propose to learn representations of mention clusters by embedding them sequentially using a recurrent neural network. Our model has no manually defined cluster features, but instead learns a global representation from the individual mentions present in each cluster. We incorporate these global representations into the mention-ranking style model from the previous section to arrive at a model that incrementally predicts a directed acyclic graph (DAG) over the mentions.

In particular, let $\hat{a}_{1:n}$ denote a sequence of antecedent predictions for each of the n mentions in a document. As noted in Section 3.1.2, these antecedent predictions induce a DAG with nodes x_n and directed edges $\{(x_n, x_{\hat{a}_n}) \mid \hat{a}_n \neq \epsilon\}$. A clustering y may furthermore be extracted from this DAG by taking the DAG's weakly connected components to be clusters. Let us accordingly define $\mathrm{wcc} : \{1, \ldots, N, \epsilon\}^N \rightarrow \{1, \ldots, N\}^N$ to be the function mapping a sequence of antecedent predictions to its implied clustering. We investigate coreference models of the form

$$\mathrm{score}(x, \hat{a}_{1:N}) = \sum_{n=1}^{N} f(x_n, \hat{a}_n) + g(x_n, \hat{a}_n, \hat{y}_{1:n-1}) \qquad (3.8)$$

where $\hat{y}_{1:n-1} = \mathrm{wcc}(\hat{a})_{1:n-1}$ and g is a global scoring function that, in making predictions for x_n, may examine (features of) the clustering $\hat{y}_{1:n-1}$ induced by the antecedent predictions $\hat{a}_{1:n-1}$, that is, the antecedent predictions made through $n-1$. The entire model, including the recurrent neural network and the mention-ranking sub-system, is trained end-to-end on the coreference task.

Experiments compare the use of learned global features to several strong baseline systems for coreference resolution. We demonstrate that the learned global representations capture important underlying information that can help resolve difficult pronominal mentions, which remain a persistent source of errors for modern coreference systems (Durrett and Klein, 2013; Kummerfeld and Klein, 2013; Wiseman et al., 2015; Martschat and Strube, 2015). Our final system improves over 0.8 points in CoNLL score over the mention-ranking system introduced in the previous chapter, setting a new state of the art (as of 2016), and the improvement is statistically significant on all three CoNLL metrics.

3.3.1 Motivating a Structured Approach

Here we motivate the use of global features for coreference resolution by focusing on the issues that may arise when resolving pronominal mentions in a purely local way. See Clark and Manning (2015) and Stoyanov and Eisner (2012) for more general motivation for using global models.

Pronoun Problems

Recent empirical work has shown that the resolution of pronominal mentions accounts for a substantial percentage of the total errors made by modern mention-ranking systems. The re-

DA: um and [I]$_1$ think that is what's - Go ahead [Linda]$_2$.

LW: Well and uh thanks goes to [you]$_1$ and to [the media]$_3$ to help [us]$_4$...So [our]$_4$ hat is off to all of [you]$_5$ as well.

Figure 3.6: A passage from the Broadcast Conversation portion of the CoNLL 2012 English development corpus (Pradhan et al., 2012). Mention are enclosed in square brackets, and co-clustered mentions are subscripted with the same number.

sults of the previous section show that on the CoNLL 2012 English development set, almost 59% of mention-ranking precision errors and almost 24% of recall errors involve pronominal mentions. Martschat and Strube (2015) found a similar pattern in their comparison of mention-ranking, mention-pair, and latent-tree models.

To see why pronouns can be so problematic, consider the passage in Figure 3.6, taken from the "Broadcast Conversation" portion of the CoNLL development set. This example is typical of Broadcast Conversation, and it is difficult because local systems learn to myopically link pronouns such as [you]$_5$ to other instances of the same pronoun that are close by, such as [you]$_1$. While this is often a reasonable strategy, in this case predicting [you]$_1$ to be an antecedent of [you]$_5$ would result in the prediction of an incoherent cluster, since [you]$_1$ is coreferent with the singular [I]$_1$, and [you]$_5$, as part of the phrase "all of you," is evidently plural. Thus, while there is enough information in the text to correctly predict [you]$_5$, doing so crucially depends on having access to the *history* of predictions made so far, and it is precisely this access to history that local models lack. More empirically, there are non-local statistical regularities involving pronouns we might hope models could exploit. For instance, in the CoNLL training data over 70% of pleonastic "it" instances and over 74% of pleonastic "you" instances follow (respectively) previous pleonastic "it" and "you" instances. Similarly, over 78% of referential "I" instances and over 68% of referential "he" instances corefer with previous “I” and “he” instances, respectively.

Accordingly, we might expect non-local models with access to global features to perform significantly better. However, models incorporating non-local features have a rather mixed track record. For instance, Björkelund and Kuhn (2014) found that cluster-level features improved their results, whereas Martschat and Strube (2015) found that they did not. Clark and Manning (2015) found that incorporating cluster-level features *beyond* those involving the pre-computed mention-pair and mention-ranking probabilities that form the basis of their agglomerative clustering coreference system did not improve performance.

Issues with Global Features

We believe a major reason for the relative ineffectiveness of global features in coreference problems is that, as noted by Clark and Manning (2015), cluster-level features can be hard to define. Specifically, it is difficult to define discrete, fixed-length features on clusters, which can be of variable size (or shape). As a result, global coreference features tend to be either too coarse or too sparse. Thus, early attempts at defining cluster-level features simply applied the coarse quantifier predicates *all*, *none*, *most* to the mention-level features defined on the mentions (or pairs of mentions) in a cluster (Culotta et al., 2007; Rahman and Ng, 2011). For example, a cluster would have the feature ‘most-female=true’ if more than half the mentions (or pairs of mentions) in the cluster have a ‘female=true’ feature.

On the other extreme, Björkelund and Kuhn (2014) define certain cluster-level features by concatenating the mention-level features of a cluster’s constituent mentions in order of the mentions’ appearance in the document. For example, if a cluster consists, in order, of the mentions (*the president*, *he*, *he*), they would define a cluster-level “type” feature ‘C-P-P=true’, which indicates that the cluster is composed, in order, of a common noun, a pronoun, and a pronoun. While very expressive, these concatenated features are often quite sparse, since clusters encountered during training can be of any size.

3.3.2 Representing Global Features

To circumvent the aforementioned issues with defining global features, we propose to learn cluster-level feature representations implicitly, by identifying the state of a (partial) cluster with the hidden state of an RNN that has consumed the sequence of mentions composing the (partial) cluster. Before providing technical details, we provide some preliminary evidence that such learned representations capture important contextual information by displaying in Figure 3.7 the learned final states of all clusters in the CoNLL development set, projected using T-SNE (van der Maaten and Hinton, 2012). Each point in the visualization represents the learned features for an entity cluster and the head words of mentions are shown for representative points. Note that the model learns to roughly separate clusters by simple distinctions such as predominant type (nominal, proper, pronominal) and number (it, they, etc), but also captures more subtle relationships such as grouping geographic terms and long strings of pronouns.

RNNs for Cluster-level Features

We use RNNs to produce feature representations of entity clusters, which will provide the basis of the global term g. Recall that we view a cluster $X^{(m)}$ as a sequence of mentions $X_1^{(m)}, \dots, X_J^{(m)}$ (ordered in linear document order). We therefore propose to embed the state(s) of $X^{(m)}$ by running an RNN over the cluster in order. In order to run an RNN over the men-

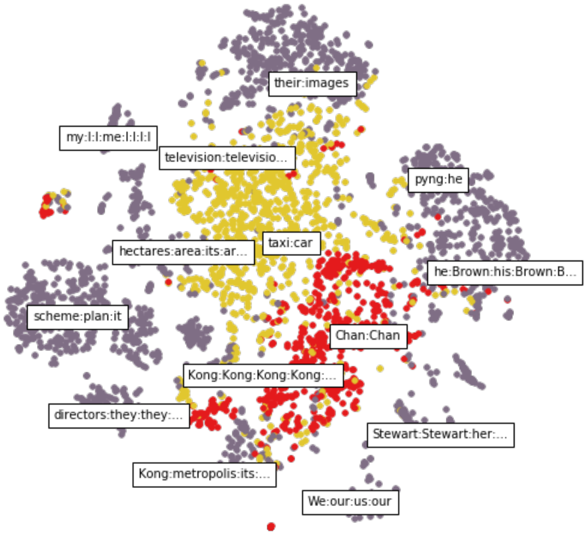


Figure 3.7: T-SNE visualization of learned entity representations on the CoNLL development set. Each point shows a gold cluster of size > 1. Yellow, red, and purple points represent predominantly common noun, proper noun, and pronoun clusters, respectively. Captions show head words of representative clusters' mentions.

tions we need an embedding function $\boldsymbol{h}_{\mathrm{c}}$ to map a mention to a real vector. As in the previous section, we produce a non-linear embeddings of a mention's features as

$$\boldsymbol{h}_{\mathrm{c}}(x_n) = \tanh(\boldsymbol{W}_{\mathrm{c}}\, \boldsymbol{\phi}_{\mathrm{a}}(x_n) + \boldsymbol{b}_{\mathrm{c}}),$$

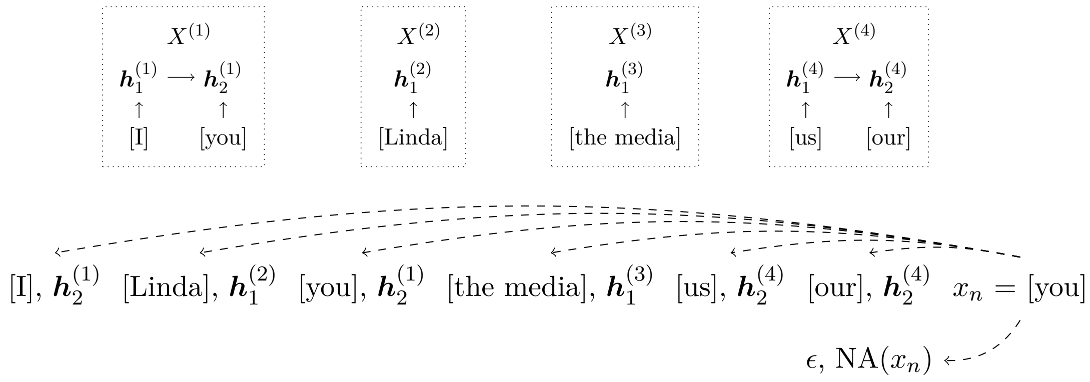
where $\boldsymbol{W}_{\mathrm{c}}$ and $\boldsymbol{b}_{\mathrm{c}}$ are parameters of the embedding. We will refer to the j'th hidden state of the RNN corresponding to $X^{(m)}$ as $\boldsymbol{h}_j^{(m)}$, and we obtain it according to the following formula

$$\boldsymbol{h}_j^{(m)} \leftarrow \mathbf{RNN}(\boldsymbol{h}_{\mathrm{c}}(X_j^{(m)}), \boldsymbol{h}_{j-1}^{(m)}; \theta),$$

assuming that $\boldsymbol{h}_0^{(m)} = \mathbf{0}$. Thus, we will effectively run an RNN over each (sequence of men-

DA: um and [I]$_1$ think that is what's - Go ahead [Linda]$_2$.

LW: Well and thanks goes to [you]$_1$ and to [the media]$_3$ to help [us]$_4$...So [our]$_4$ hat is off to all of [you]$_5$...



$X^{(1)}$
$\boldsymbol{h}_1^{(1)} \longrightarrow \boldsymbol{h}_2^{(1)}$
[I]
[you]
$X^{(2)}$
$\boldsymbol{h}_1^{(2)}$
[Linda]
$X^{(3)}$
$\boldsymbol{h}_1^{(3)}$
[the media]
$X^{(4)}$
$\boldsymbol{h}_1^{(4)} \longrightarrow \boldsymbol{h}_2^{(4)}$
[us]
[our]
[I], $\boldsymbol{h}_2^{(1)}$
[Linda], $\boldsymbol{h}_1^{(2)}$
[you], $\boldsymbol{h}_2^{(1)}$
[the media], $\boldsymbol{h}_1^{(3)}$
[us], $\boldsymbol{h}_2^{(4)}$
[our], $\boldsymbol{h}_2^{(4)}$
$x_n =$ [you]
ϵ, $\mathrm{NA}(x_n)$

Figure 3.8: Full RNN example for handling the mention x_n = [you]. There are currently four entity clusters in scope $X^{(1)}, X^{(2)}, X^{(3)}, X^{(4)}$ based on unseen previous decisions (y). Each cluster has a corresponding RNN state, two of which ($\boldsymbol{h}^{(1)}$ and $\boldsymbol{h}^{(4)}$) have processed multiple mentions (with $X^{(1)}$ notably including a singular mention [I]). At the bottom, we show the complete mention-ranking process. Each previous mention is considered as an antecedent, and the global term considers the antecedent clusters' current hidden state. Selecting ϵ is treated with a special case $\mathrm{NA}(x_n)$.

tions corresponding to a) cluster $X^{(m)}$ in the document, and thereby generate a hidden state $\boldsymbol{h}_j^{(m)}$ corresponding to each step of each cluster in the document. Concretely, this can be implemented by maintaining M RNNs – one for each cluster – that all share the parameters θ. The process is illustrated in the top portion of Figure 3.8. We now describe how the RNN defined above is used within an end-to-end coreference system.

Full Model

Recall that our inference objective is to maximize the score of both a local mention ranking term as well as a global term based on the current clusters, as in Equation (3.8). There, the local score $f(x_n, a)$ is computed as in the previous section. The global scoring function g, on the other hand, uses the history of previous decisions. Define $\boldsymbol{h}_{<n}^{(m)}$ as the hidden state of cluster m before a decision is made for x_n – that is, $\boldsymbol{h}_{<n}^{(m)}$ is the state of cluster m's RNN after it has consumed all mentions in the cluster *preceding* x_n. We define g as

$$g(x_n, a, \hat{a}_{1:n-1}) = \begin{cases} \boldsymbol{h}_{\mathrm{c}}(x_n)^{\mathsf{T}} \boldsymbol{h}_{<n}^{(y_a)} & \text{if } a \neq \epsilon \\ \mathrm{NA}(x_n) & \text{if } a = \epsilon, \end{cases}$$

where NA gives a score for assigning ϵ based on a non-linear function of all of the current hidden states:

$$\mathrm{NA}(x_n) = \boldsymbol{q}^{\mathsf{T}} \tanh\left(\boldsymbol{W}_s \begin{bmatrix} \boldsymbol{\phi}_{\mathrm{a}}(x_n) \\ \sum_{m=1}^{M} \boldsymbol{h}_{<n}^{(m)} \end{bmatrix} + \boldsymbol{b}_s\right).$$

See Figure 3.8 for a diagram. The intuition behind the first case in g is that in considering whether a is a good antecedent for x_n, we add a term to the score that examines how well x_n matches with the mentions already in $X^{(y_a)}$, the cluster associated with a. This matching score is expressed via a dot-product. In the second case, when predicting that x_n is non-anaphoric, we add the NA term to the score, which examines the (sum of) the current states $\boldsymbol{h}_{<n}^{(m)}$ of all clusters. This information is useful both because it allows the non-anaphoric score to incorporate information about potential antecedents, and because the occurrence of certain singleton-clusters often predicts the occurrence of future singleton-clusters.

An Alternate View

While we have largely motivated the aforementioned model and prediction scheme from the perspective of capturing global features useful for coreference resolution, it is worth emphasizing that this approach also represents an instance of the search-based structured prediction setup advocated in this thesis. In particular, the prediction of a coreference DAG is decomposed into a sequence of incremental antecedent predictions, which condition on the history of the predictions made using an RNN.

Training

The whole system is trained end-to-end on coreference using backpropagation. For a given training document, let y be the oracle mapping from mention to cluster, which induces an oracle clustering. As noted in the previous section, while at training time we do have oracle clusters, we do not have oracle antecedents, and so we again train with a latent-variable, slack-rescaled objective:

$$\sum_{n=1}^{N} \max_{a \in \{1,\ldots,n-1,\epsilon\}} \Delta(x_n, a)(1 + f(x_n, a) + g(x_n, a, y) - f(x_n, y_n^\ell) - g(x_n, y_n^\ell, y)),$$

where the latent antecedent y_n^ℓ is defined as in Equation (3.5), and the Δ terms as in Equations (3.7).

Note that in training we use the oracle clusters y. Since these are known a priori, we can pre-compute all the hidden states $\boldsymbol{h}_j^{(m)}$ in a document, which makes training quite simple and efficient. This approach contrasts in particular with the work of Björkelund and Kuhn (2014) — who also incorporate global information in mention-ranking — in that they train against latent *trees*, which are not annotated and must be searched for during training. On the other hand, training on oracle clusters leads to a mismatch between training and test, which can hurt performance.

Search

When moving from a strictly local objective to one with global features, the test-time search problem becomes intractable. The local objective requires $O(N^2)$ time, whereas the full clustering problem is NP-Hard. Past work with global features has used integer linear programming solvers for exact search (Chang et al., 2013; Peng et al., 2015), or beam search with (delayed) early update training for an approximate solution (Björkelund and Kuhn, 2014). In contrast, we simply use greedy search at test time, which also requires $O(N^2)$ time. While beam search is a natural way to decrease search error at test time, it may fail to help if training involves a *local* margin objective (as in our case), since scores need not be calibrated across local decisions. We accordingly attempted to train various locally normalized versions of our

Algorithm 4 Greedy search with global RNNs

```
1: procedure GreedyCluster($(x_1, \dots, x_N)$)
2:     Initialize clusters $X^{(1)} \dots$ as empty lists, hidden states $\boldsymbol{h}^{(0)}, \dots$ as $\mathbf{0}$ vectors in $\mathbb{R}^D$, $\hat{y}$ as map from mention to cluster, and cluster counter $M \leftarrow 0$
3:     for $n = 2 \dots N$ do
4:         $\hat{a} \leftarrow \underset{a \in \{1,\dots,n-1,\epsilon\}}{\arg\max} f(x_n, a) + g(x_n, a, \hat{y}_{1:n-1})$
5:         $m \leftarrow \hat{y}_{\hat{a}}$
6:         if $\hat{a} = \epsilon$ then
7:             $M \leftarrow M + 1$
8:             $m \leftarrow M$
9:         append $x_n$ to $X^{(m)}$
10:        $\hat{y}_n \leftarrow m$
11:        $\boldsymbol{h}^{(m)} \leftarrow \mathbf{RNN}(\boldsymbol{h}_{\mathrm{c}}(x_n), \boldsymbol{h}^{(m)})$
12:    return $X^{(1)}, \dots, X^{(M)}$
```

model, but found that they underperformed. We also experimented with training approaches and model variants that expose the model to its own predictions (Daumé III et al., 2009; Ross et al., 2011; Bengio et al., 2015), but found that these yielded a fairly small performance improvement, at the cost of a more complicated training scheme. The full algorithm is shown in Algorithm 4. The greedy search algorithm is identical to a simple mention-ranking system, with the exception of line 11, which updates the current RNN representation based on the previous decision that was made, and line 4, which then uses this cluster representation as part of scoring.

3.3.3 Experiments

We run experiments on the CoNLL 2012 English shared task (Pradhan et al., 2012). The task uses the OntoNotes corpus (Hovy et al., 2006), consisting of 3,493 documents in various domains and formats. We use the experimental split provided in the shared task. For all experiments, we use the Berkeley Coreference System (Durrett and Klein, 2013) for mention extraction and to compute features $\boldsymbol{\phi}_{\mathrm{a}}$ and $\boldsymbol{\phi}_{\mathrm{p}}$.

FEATURES We use the raw BASIC+ feature sets in Figure 3.2, with the following modifications:

- We remove all features from $\boldsymbol{\phi}_{\mathrm{p}}$ that concatenate a feature of the antecedent with a feature of the current mention, such as bi-head features.
- We add true-cased head features, a current speaker indicator feature, and a 2-character genre (out of {bc,bn,mz,nw,pt,tc,wb}) indicator to $\boldsymbol{\phi}_{\mathrm{p}}$ and $\boldsymbol{\phi}_{\mathrm{a}}$.
- We add features indicating if a mention has a substring overlap with the current speaker ($\boldsymbol{\phi}_{\mathrm{p}}$ and $\boldsymbol{\phi}_{\mathrm{a}}$), and if an antecedent has a substring overlap with a speaker distinct from the current mention's speaker ($\boldsymbol{\phi}_{\mathrm{p}}$).
- We add a single centered, rescaled document position feature to each mention when learning $\boldsymbol{h}_{\mathrm{c}}$. We calculate a mention x_n's rescaled document position as $\frac{2n-N-1}{N-1}$.

These modifications result in there being approximately 14K distinct features in $\boldsymbol{\phi}_{\mathrm{a}}$ and approximately 28K distinct features in $\boldsymbol{\phi}_{\mathrm{p}}$, which is far fewer features than has been typical in past work.

For training, we use document-size minibatches, which allows for efficient pre-computation of RNN states, and we minimize the loss with AdaGrad (Duchi et al., 2011) (after clipping LSTM gradients to lie (elementwise) in $(-10, 10)$). We find that the initial learning rate chosen for AdaGrad has a significant impact on results, and we choose learning rates for each layer out of $\{0.1, 0.02, 0.01, 0.002, 0.001\}$.

In experiments, we set $\boldsymbol{h}_{\mathrm{a}}(x_n)$, $\boldsymbol{h}_{\mathrm{c}}(x_n)$, and $\boldsymbol{h}^{(m)}$ to be $\in \mathbb{R}^{200}$, and $\boldsymbol{h}_{\mathrm{p}}(x_n, y) \in \mathbb{R}^{700}$. We use a single-layer LSTM (without "peep-hole" connections), as implemented in the `element-rnn` library (Léonard et al., 2015). For regularization, we apply Dropout (Srivastava et al., 2014) with a rate of 0.4 before applying the linear weights $\boldsymbol{u}$, and we also apply Dropout with a rate of 0.3 to the LSTM states before forming the dot-product scores. We use the same cost-weights $\boldsymbol{\alpha}$ and pretraining described in the previous section. For final results, we train on both training and development portions of the CoNLL data. Scoring uses the official

CoNLL 2012 script (Pradhan et al., 2014; Luo et al., 2014). Code for our system is available at `https://github.com/swiseman/nn_coref`.

Results

System	MUC			B^3			CEAF$_e$			
	P	R	F$_1$	P	R	F$_1$	P	R	F$_1$	CoNLL
B&K	74.3	67.46	70.72	62.71	54.96	58.58	59.4	52.27	55.61	61.63
M&S	76.72	68.13	72.17	66.12	54.22	59.58	59.47	52.33	55.67	62.47
C&M	76.12	69.38	72.59	65.64	56.01	60.44	59.44	52.98	56.02	63.02
Peng	-	-	72.22	-	-	60.50	-	-	56.37	63.03
(3.2), $\boldsymbol{g}_1$	76.23	69.31	72.60	66.07	55.83	60.52	59.41	54.88	57.05	63.39
(3.8)	77.49	69.75	**73.42**	66.83	56.95	**61.50**	62.14	53.85	**57.70**	**64.21**

Table 3.7: Results on CoNLL 2012 English test set. We compare against a more recent crop of state-of-the-art systems, including (in order) Bjorkelund and Kuhn (2014), Martschat and Strube (2015), Clark and Manning (2015), Peng et al. (2015), and the results from the previous section. F$_1$ gains are significant ($p < 0.05$ under the bootstrap resample test (Koehn, 2004)) compared with those of the previous section for all metrics.

In Table 3.7 we present our main results on the CoNLL English test set, and compare with other recent state-of-the-art systems. We see a statistically significant improvement of over 0.8 CoNLL points over the previous state of the art, and the highest F$_1$ scores to date on all three CoNLL metrics.

We now consider in more detail the impact of global features and RNNs on performance. For these experiments, we report MUC, B^3, and CEAF$_e$ F$_1$-scores in Table 3.8 as well as errors broken down by mention type and by whether the mention is anaphoric or not in Table 3.9. Table 3.9 further partitions errors into FL, FN, and WL categories, which are defined in Section 3.3.2. We typically think of FL and WL as representing precision errors, and FN as representing recall errors.

Our experiments consider several different settings. First, we consider an oracle setting ("RNN, OH" in tables), in which the model receives $y_{1:n-1}$, the oracle partial clustering of

	MUC	B^3	CEAF_e	CoNLL
MR	73.06	62.66	58.98	64.90
Avg, OH	73.30	63.06	58.85	65.07
RNN, GH	73.63	63.23	59.56	65.47
RNN, OH	74.26	63.89	59.54	65.90

Table 3.8: F_1 scores of models described in text on CoNLL 2012 development set. Rows in grey highlight models using oracle history.

	Non-Anaphoric (FL)		
	Nom. HM	Nom. No HM	Pron.
MR	1061	130	1075
Avg, OH	983	140	1011
RNN, GH	914	125	893
RNN, OH	913	130	842
# Mentions	9.0K	22.2K	3.1K
	Anaphoric (FN + WL)		
Model	Nom. HM	Nom. No HM	Pron.
MR	665+326	666+56	533+796
Avg, OH	781+300	641+60	578+744
RNN, GH	767+303	648+57	664+727
RNN, OH	750+289	648+52	611+686
# Mentions	4.7K	1.0K	7.3K

Table 3.9: Number of ``false link'' (fl) errors on non-anaphoric mentions (top) and number of ``false new'' (fn) and ``wrong link'' (wl) errors on anaphoric mentions (bottom) on CoNLL 2012 development set. Mentions are categorized as nominal or proper with (previous) head match (Nom. HM), nominal or proper with no head match (Nom. No HM), and pronominal. Models are described in the text, and rows in grey highlight models using oracle history.

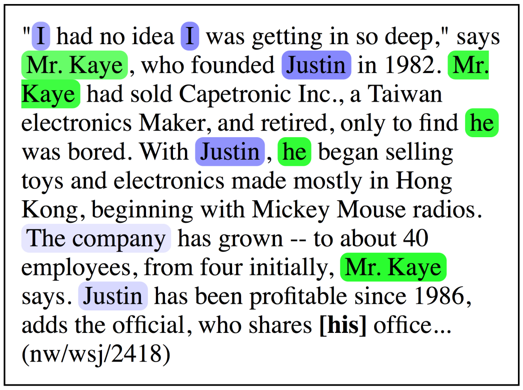
all mentions preceding x_n in the document, and is therefore not forced to rely on its own past predictions when predicting x_n. This provides us with an upper bound on the performance achievable with our model. Next, we consider the performance of the model under a greedy inference strategy (RNN, GH), as in Algorithm 4. Finally, for baselines we consider the mention-ranking system in the previous section using our updated feature-set, as well as a non-local baseline with oracle history (Avg, OH), which averages the representations $\boldsymbol{h}_{\text{c}}(x_j)$ for all $x_j \in X^{(m)}$, rather than feed them through an RNN; errors are still backpropagated through the $\boldsymbol{h}_{\mathrm{c}}$ representations during learning.

In Table 3.9 we see that the RNN improves performance overall, with the most dramatic improvements on non-anaphoric pronouns, though errors are also decreased significantly for non-anaphoric nominal and proper mentions that follow at least one mention with the same head. While WL errors also decrease for both these mention-categories under the RNN model, FN errors increase. Importantly, the RNN performance is significantly better than that of the Avg baseline, which barely improves over mention-ranking, even with oracle history. This suggests that modeling the sequence of mentions in a cluster is advantageous. We also note that while RNN performance degrades in both precision and recall when moving from the oracle history upper-bound to a greedy setting, we are still able to recover a significant portion of the possible performance improvement.

Qualitative Analysis

In this section we consider in detail the impact of the g term in the RNN scoring function on the two error categories that improve most under the RNN model (as shown in Table 3.9), namely, pronominal WL errors and pronominal FL errors. We consider an example from the CoNLL development set in each category on which the baseline MR model makes an error but the greedy RNN model does not.

The example in Figure 3.9 involves the resolution of the ambiguous pronoun "his," which is bracketed and in bold in the figure. Whereas the baseline MR model *incorrectly* predicts "his" to corefer with the closest gender-consistent antecedent "Justin" — thus making a WL error — the greedy RNN model correctly predicts "his" to corefer with "Mr. Kaye" in the previous sentence. (Note that "the official" also refers to Mr. Kaye). To get a sense of the greedy RNN

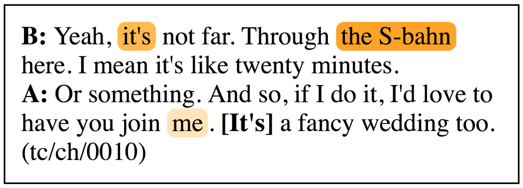


"I had no idea I was getting in so deep," says Mr. Kaye, who founded Justin in 1982. Mr. Kaye had sold Capetronic Inc., a Taiwan electronics Maker, and retired, only to find he was bored. With Justin, he began selling toys and electronics made mostly in Hong Kong, beginning with Mickey Mouse radios. The company has grown -- to about 40 employees, from four initially, Mr. Kaye says. Justin has been profitable since 1986, adds the official, who shares **[his]** office... (nw/wsj/2418)

Figure 3.9: Cluster predictions of greedy RNN model; co-clustered mentions are of the same color, and intensity of mention x_j corresponds to $\boldsymbol{h}_c(x_n)^\top \boldsymbol{h}^{(i)}_{<k}$, where $k = j + 1, i \in \{1, 2\}$, and x_n = ``his.'' See text for full description.

model's decision-making on this example, we color the mentions the greedy RNN model has predicted to corefer with "Mr. Kaye" in green, and the mentions it has predicted to corefer with "Justin" in blue. (Note that the model incorrectly predicts the initial "I" mentions to corefer with "Justin.") Letting $X^{(1)}$ refer to the blue cluster, $X^{(2)}$ refer to the green cluster, and x_n refer to the ambiguous mention "his," we further shade each mention x_j in $X^{(1)}$ so that its intensity corresponds to $\boldsymbol{h}_c(x_n)^\top \boldsymbol{h}^{(1)}_{<k}$, where $k = j + 1$; mentions in $X^{(2)}$ are shaded analogously. Thus, the shading shows how highly g scores the compatibility between "his" and a cluster $X^{(i)}$ as each of $X^{(i)}$'s mentions is added. We see that when the initial "Justin" mentions are added to $X^{(1)}$ the g-score is relatively high. However, after "The company" is correctly predicted to corefer with "Justin," the score of $X^{(1)}$ drops, since companies are generally not coreferent with pronouns like "his."

Figure 3.10 shows an example (consisting of a telephone conversation between "A" and "B") in which the bracketed pronoun "It's" is being used pleonastically. Whereas the baseline MR model predicts "It's" to corefer with a previous "it" — thus making a FL error — the greedy RNN model does not. In Figure 3.10 the final mention in three preceding clusters



B: Yeah, it's not far. Through the S-bahn
here. I mean it's like twenty minutes.
A: Or something. And so, if I do it, I'd love to
have you join me . **[It's]** a fancy wedding too.
(tc/ch/0010)

Figure 3.10: Magnitudes of gradients of NA score applied to bold ``It's'' with respect to final mention in three preceding clusters. See text for full description.

is shaded so its intensity corresponds to the magnitude of the gradient of the NA term in g with respect to that mention. This visualization resembles the "saliency" technique of Li et al. (2016), and it attempts to gives a sense of the contribution of a (preceding) cluster in the calculation of the NA score.

We see that the potential antecedent "S-Bahn" has a large gradient, but also that the initial, obviously pleonastic use of "it's" has a large gradient, which may suggest that earlier, easier predictions of pleonasm can inform subsequent predictions.

3.3.4 Conclusion

We have presented a neural, structured model for coreference resolution which proceeds by incrementally predicting edges in a DAG, embedding the history of previous antecedent predictions using an RNN. We have seen that this approach outperforms the neural, unstructured mention-ranking model in the previous section. Moreover, as depicted in Figure 3.9, this approach gives us a dynamic representation of the entities in a document as each mention is added to a cluster.

4

Structured Training of Sequence-to-Sequence Models[1]

4.1 Introduction

In this chapter we address several structured prediction-related issues with the now-standard Sequence-to-Sequence model. We will suggest using techniques from search-based structured prediction (again with RNNs) to address these issues. Sequence-to-Sequence learning (Sutskever

[1] The material in this chapter is adapted from Wiseman and Rush (2016).

et al., 2011, 2014) (herein, seq2seq) has rapidly become a very useful and surprisingly general-purpose tool for natural language processing. In addition to demonstrating impressive results for machine translation (Bahdanau et al., 2015), roughly the same model and training have also proven to be useful for sentence compression (Filippova et al., 2015), parsing (Vinyals et al., 2015), and dialogue systems (Serban et al., 2016), and they additionally underlie other text generation applications, such as image or video captioning (Venugopalan et al., 2015; Xu et al., 2015).

The dominant approach to training a seq2seq system is as a conditional language model, with training maximizing the likelihood of each successive target word conditioned on the input sequence and the *gold* history of target words. Thus, training uses a strictly word-level loss, usually cross-entropy over the target vocabulary. This approach has proven to be very effective and efficient for training neural language models, and seq2seq models similarly obtain impressive perplexities for word-generation tasks.

Notably, however, seq2seq models are not used as conditional language models at test-time; they must instead generate fully-formed word sequences. In practice, generation is accomplished by searching over output sequences greedily or with beam search. In this context, Ranzato et al. (2016) note that the combination of the training and generation scheme just described leads to at least two major issues:

1. *Exposure Bias*: the model is never exposed to its own errors during training, and so the inferred histories at test-time do not resemble the gold training histories.
2. *Loss-Evaluation Mismatch*: training uses a word-level loss, while at test-time we target improving sequence-level evaluation metrics, which do not decompose over words, such as BLEU (Papineni et al., 2002).

We might additionally add the concern of *label bias* (Lafferty et al., 2001) to the list, since word-probabilities at each time-step are locally normalized, guaranteeing that successors of incorrect histories receive the same mass as do the successors of the true history. Thus, seq2seq models suffer from all the issues identified with MEMMs in Section 2.2.1, in addition to the concern of Loss-Evaluation mismatch.

In this chapter we develop a non-probabilistic variant of the seq2seq model that can assign a score to any possible target *sequence*, and we propose a training procedure, inspired by the learning as search optimization (LaSO) framework of Daumé III and Marcu (2005), that defines a loss function in terms of errors made during beam search. Furthermore, we provide an efficient algorithm to back-propagate through the beam-search procedure during seq2seq training. This approach offers a possible solution to each of the three aforementioned issues, while largely maintaining the model architecture and training efficiency of standard seq2seq learning. Moreover, by scoring sequences rather than words, our approach also allows for enforcing hard-constraints on sequence generation *at training time*. To test out the effectiveness of the proposed approach, we develop a general-purpose seq2seq system with beam search optimization. We run experiments on three very different problems: word ordering, syntactic parsing, and machine translation, and compare to a highly-tuned seq2seq system with attention (Luong et al., 2015). The version with beam search optimization shows significant improvements on all three tasks, and particular improvements on tasks that require difficult search.

4.2 Background

In the simplest seq2seq scenario, we are given a collection of source-target sequence pairs and tasked with learning to generate target sequences from source sequences. For instance, we might view machine translation in this way, where in particular we attempt to generate English sentences from (corresponding) French sentences. Seq2seq models are part of the broader class of "encoder-decoder" models (Cho et al., 2014), which first use an encoding model to transform an input object x into an encoded representation, which we will denote with $\boldsymbol{x}$. Many different sequential (and non-sequential) encoders have proven to be effective for different source domains. In this work we are agnostic to the form of the encoding model, and simply some way of encoding the input x.

Once the input sequence is encoded, seq2seq models generate a target sequence using a *decoder*. The decoder is tasked with generating a target sequence of words from a target vocabulary $\mathcal{V}$. In particular, words are generated sequentially by conditioning on the input representation $\boldsymbol{x}$ and on the previously generated words or *history*. We use the notation $w_{1:T}$ to refer to an arbitrary word sequence of length T, and the notation $y_{1:T}$ to refer to the *gold* (i.e., correct) target word sequence for an input x. Many seq2seq systems utilize a recurrent neural network (RNN) for the decoder model, and we focus on these scenarios in this chapter.

RNN decoders are typically trained to act as conditional language models. That is, one attempts to model the probability of the t'th target word conditioned on $\boldsymbol{x}$ and the target history by stipulating that

$$p(w_t | w_{1:t-1}, \boldsymbol{x}) = g(w_t, \boldsymbol{h}_{t-1}, \boldsymbol{x}), \tag{4.1}$$

for some parameterized function g typically computed with an affine layer followed by a softmax. In computing these probabilities, the state $\boldsymbol{h}_{t-1}$ represents the target history, and $\boldsymbol{h}_0$ is typically set to be some function of $\boldsymbol{x}$. The complete model (including encoder) is trained, analogously to a neural language model, to minimize the cross-entropy loss at each time-step while conditioning on the gold history in the training data. That is, the model is trained to minimize

$$-\ln\prod_{t=1}^{T} p(y_t|y_{1:t-1}, \boldsymbol{x}). \tag{4.2}$$

We note that this model and training style closely resemble the MEMMs discussed in Section 2.2.1.

Once the decoder is trained, discrete sequence generation can be performed by approximately maximizing the probability of the target sequence under the conditional distribution:

$$\hat{y}_{1:T} = \mathrm{argbeam}_{w_{1:T}} \prod_{t=1}^{T} p(w_t|w_{1:t-1}, \boldsymbol{x}),$$

where we use the notation argbeam to emphasize that the decoding process requires heuristic search, since the RNN model is non-Markovian. In practice, a simple beam search procedure that explores K prospective histories at each time-step has proven to be an effective decoding approach. However, as noted above, decoding in this manner after conditional language-model style training potentially suffers from the issues of exposure bias and label bias, which motivates the work of this paper.

4.3 Beam Search Optimization

We begin by making one small change to the seq2seq modeling framework. Instead of predicting the probability of the next word, we instead learn to produce (non-probabilistic) scores for ranking sequences. Define the score of a sequence consisting of *history* $w_{1:t-1}$ followed by a single word w_t as $f(w_t, \boldsymbol{h}_{t-1}, \boldsymbol{x})$, where f is a parameterized function examining the current hidden-state of the relevant RNN at time $t-1$ as well as the input representation $\boldsymbol{x}$. In experiments, our f will have an identical form to g but *without* the final softmax transformation (which transforms unnormalized scores into probabilities), thereby allowing the model to avoid issues associated with the label bias problem.

More importantly, we also modify how this model is trained. Ideally we would train by comparing the gold sequence to the highest-scoring complete sequence. However, because finding the argmax sequence according to this model is intractable, we propose to adopt a LaSO-like (Daumé III and Marcu, 2005) scheme to train, which we will refer to as beam search optimization (BSO). In particular, we define a loss that penalizes the gold sequence falling off the beam during training.[2] The proposed training approach is a simple way to expose the model to incorrect histories and to match the training procedure to test generation. Furthermore we show that it can be implemented efficiently without changing the asymptotic run-time of training, beyond a factor of the beam size K.

4.3.1 Search-Based Loss

We now formalize this notion of a search-based loss for RNN training. Assume we have a set S_t of K candidate sequences of length t. We can calculate a score for each sequence in S_t using a scoring function f parameterized with an RNN, as above, and we define the sequence $\hat{y}_{1:t}^{(K)} \in S_t$ to be the K'th ranked sequence in S_t according to f. That is, assuming distinct scores,

$$|\{\hat{y}_{1:t}^{(k)} \in S_t \mid f(\hat{y}_t^{(k)}, \hat{\boldsymbol{h}}_{t-1}^{(k)}) > f(\hat{y}_t^{(K)}, \hat{\boldsymbol{h}}_{t-1}^{(K)})\}| = K - 1,$$

[2] Using a non-probabilistic model further allows us to incur no loss (and thus require no update to parameters) when the gold sequence *is* on the beam; this contrasts with models based on a CRF loss, such as those of Andor et al. (2016) and Zhou et al. (2015), though in training those models are simply not updated when the gold sequence remains on the beam.

where $\hat{y}_t^{(k)}$ is the t'th token in $\hat{y}_{1:t}^{(k)}$, $\hat{\boldsymbol{h}}_{t-1}^{(k)}$ is the RNN state corresponding to its $t-1$'st step, and where we have omitted the $\boldsymbol{x}$ argument to f for brevity.

We now define a loss function that gives loss each time the score of the gold prefix $y_{1:t}$ does not exceed that of $\hat{y}_{1:t}^{(K)}$ by a margin:

$$L(\theta) = \sum_{t=1}^{T} \Delta(\hat{y}_{1:t}^{(K)}) \left[1 - f(y_t, \boldsymbol{h}_{t-1}) + f(\hat{y}_t^{(K)}, \hat{\boldsymbol{h}}_{t-1}^{(K)}) \right].$$

Above, θ refers to the set of all parameters, and the $\Delta(\hat{y}_{1:t}^{(K)})$ term denotes a mistake-specific cost-function, which allows us to scale the loss depending on the severity of erroneously predicting $\hat{y}_{1:t}^{(K)}$; it is assumed to return 0 when the margin requirement is satisfied, and a positive number otherwise. It is this term that allows us to use sequence- rather than word-level costs in training (addressing the 2nd issue in the introduction). For instance, when training a seq2seq model for machine translation, it may be desirable to have $\Delta(\hat{y}_{1:t}^{(K)})$ be inversely related to the partial sentence-level BLEU score of $\hat{y}_{1:t}^{(K)}$ with $y_{1:t}$; we experiment along these lines in Section 4.4.3. Finally, because we want the full gold sequence to be at the top of the beam at the end of search, when $t = T$ we modify the loss to require the score of $y_{1:T}$ to exceed the score of the *highest* ranked incorrect prediction by a margin.

We can optimize the loss L using a two-step process: (1) in a forward pass, we compute candidate sets S_t and record margin violations (sequences with non-zero loss); (2) in a backward pass, we back-propagate the errors through the seq2seq RNNs. Unlike standard seq2seq training, the first-step requires running search (in our case beam search) to find margin violations. The second step can be done by adapting back-propagation through time (BPTT). We next discuss the details of this process.

4.3.2 Forward: Find Violations

In order to minimize this loss, we need to specify a procedure for constructing candidate sequences $\hat{y}^{(k)}_{1:t}$ at each time step t so that we find margin violations. We follow LaSO (rather than early-update[3]; see Section 2.3.2) and build candidates in a recursive manner. If there was no margin violation at $t{-}1$, then S_t is constructed using a standard beam search update. If there was a margin violation, S_t is constructed as the K best sequences assuming the gold history $y_{1:t-1}$ through time-step $t{-}1$.

Formally, assume the function succ maps a sequence $w_{1:t-1} \in \mathcal{V}^{t-1}$ to the set of all valid sequences of length t that can be formed by appending to it a valid word $w \in \mathcal{V}$. In the simplest, unconstrained case, we will have

$$\mathrm{succ}(w_{1:t-1}) = \{w_{1:t-1}, w \mid w \in \mathcal{V}\}.$$

As an important aside, note that for some problems it may be preferable to define a succ function which imposes hard constraints on successor sequences. For instance, if we would like to use seq2seq models for parsing (by emitting a constituency or dependency structure encoded into a sequence in some way), we will have hard constraints on the sequences the model can output, namely, that they represent valid parses. While hard constraints such as these would be difficult to add to standard seq2seq at training time, in our framework they can naturally be added to the succ function, allowing us to *train* with hard constraints; we experiment along these lines in Section 4.4.3, where we refer to a model trained with constrained

[3] We found that training with early-update rather than (delayed) LaSO did not work well, even after pre-training. Given the success of early-update in many NLP tasks this was somewhat surprising. We leave this question to future work.

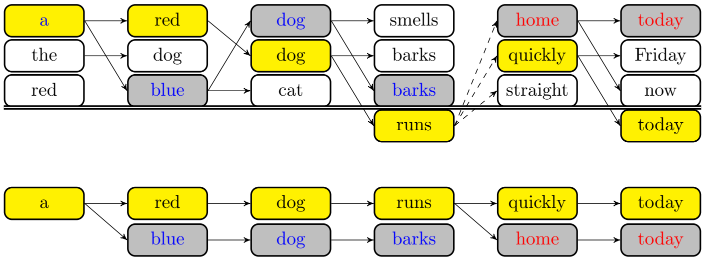


Figure 4.1: Top: possible $\hat{y}^{(k)}_{1:t}$ formed in training with a beam of size $K = 3$ and with gold sequence $y_{1:6}$ = ``a red dog runs quickly today''. The gold sequence is highlighted in yellow, and the predicted prefixes involved in margin violations (at $t = 4$ and $t = 6$) are in gray. Note that time-step $T = 6$ uses a different loss criterion. Bottom: prefixes that actually participate in the loss, arranged to illustrate the back-propagation process.

beam search as ConBSO.

Having defined an appropriate succ function, we specify the candidate set as:

$$S_t = \text{topK} \begin{cases} \text{succ}(y_{1:t-1}) & \text{violation at } t{-}1 \\ \bigcup_{k=1}^{K} \text{succ}(\hat{y}^{(k)}_{1:t-1}) & \text{otherwise,} \end{cases}$$

where we have a margin violation at $t{-}1$ iff $f(y_{t-1}, \boldsymbol{h}_{t-2}) < f(\hat{y}^{(K)}_{t-1}, \hat{\boldsymbol{h}}^{(K)}_{t-2}) + 1$, and where topK considers the scores given by f. This search procedure is illustrated in the top portion of Figure 4.1. In the forward pass of our training algorithm, shown as the first part of Algorithm 5, we run this version of beam search and collect all sequences and their hidden states that lead to losses.

4.3.3 Backward: Merge Sequences

Once we have collected margin violations we can run backpropagation to compute parameter updates. Assume a margin violation occurs at time-step t between the predicted history $\hat{y}^{(K)}_{1:t}$ and the gold history $y_{1:t}$. As in standard seq2seq training we must back-propagate this error through the gold history; however, unlike seq2seq we also have a gradient for the wrongly predicted history.

Recall that to back-propagate errors through an RNN we run a recursive backward procedure — denoted below by **BRNN** — at each time-step t, which accumulates the gradients of next-step and future losses with respect to $\boldsymbol{h}_t$. We have:

$$\nabla_{\boldsymbol{h}_t} L \leftarrow \mathbf{BRNN}(\nabla_{\boldsymbol{h}_t} L_{t+1}, \nabla_{\boldsymbol{h}_{t+1}} L),$$

where L_{t+1} is the loss at step $t+1$, deriving, for instance, from the score $f(y_{t+1}, \boldsymbol{h}_t)$. Running this **BRNN** procedure from $t = T-1$ to $t = 0$ is known as back-propagation through time (BPTT).

In determining the total computational cost of back-propagation here, first note that in the worst case there is one violation at each time-step, which leads to T independent, incorrect sequences. Since we need to call **BRNN** $O(T)$ times for each sequence, a naive strategy of running BPTT for each incorrect sequence would lead to an $O(T^2)$ backward pass, rather than the $O(T)$ time required for the standard seq2seq approach. Instead, it is possible to avoid a quadratic backward pass by sharing **BRNN** operations. Even better, our use of the LaSO beam-update during training-time search makes the backward pass independent of K, the size of the beam, as well.

We informally illustrate the process in Figure 4.1. The top of the diagram shows a possible sequence of $\hat{y}_{1:t}^{(k)}$ formed during search with a beam of size 3 for the target sequence $y =$ "a red dog runs quickly today." When the gold sequence falls off the beam at $t = 4$, search resumes with $S_5 = \text{succ}(y_{1:4})$, and so all subsequent predicted sequences have $y_{1:4}$ as a prefix and are thus functions of $\boldsymbol{h}_4$. Moreover, because our loss function only involves the scores of the gold

Algorithm 5 Seq2seq Beam-Search Optimization

```
procedure BSO($\boldsymbol{x}, K_{tr}$, succ)
    /*FORWARD*/
    Init empty storage $\hat{y}_{1:T}$ and $\hat{\boldsymbol{h}}_{1:T}$; init $S_1$
    $r \leftarrow 0$; $violations \leftarrow \{0\}$
    for $t = 1, \ldots, T$ do
        $K = K_{tr}$ if $t \neq T$ else $\operatorname*{arg\,max}_{k:\hat{y}^{(k)}_{1:t} \neq y_{1:t}} f(\hat{y}^{(k)}_t, \hat{\boldsymbol{h}}^{(k)}_{t-1})$
        if $f(y_t, \boldsymbol{h}_{t-1}) < f(\hat{y}^{(K)}_t, \hat{\boldsymbol{h}}^{(K)}_{t-1}) + 1$ then
            $\hat{\boldsymbol{h}}_{r:t-1} \leftarrow \hat{\boldsymbol{h}}^{(K)}_{r:t-1}$
            $\hat{y}_{r+1:t} \leftarrow \hat{y}^{(K)}_{r+1:t}$
            Add $t$ to $violations$
            $r \leftarrow t$
            $S_{t+1} \leftarrow$ topK(succ($y_{1:t}$))
        else
            $S_{t+1} \leftarrow$ topK($\bigcup_{k=1}^{K}$ succ($\hat{y}^{(k)}_{1:t}$))
    /*BACKWARD*/
    $grad\_\boldsymbol{h}_T \leftarrow \mathbf{0}$; $grad\_\widehat{\boldsymbol{h}}_T \leftarrow \mathbf{0}$
    for $t = T - 1, \ldots, 1$ do
        $grad\_\boldsymbol{h}_t \leftarrow \mathbf{BRNN}(\nabla_{\boldsymbol{h}_t} L_{t+1}, grad\_\boldsymbol{h}_{t+1})$
        $grad\_\widehat{\boldsymbol{h}}_t \leftarrow \mathbf{BRNN}(\nabla_{\widehat{\boldsymbol{h}}_t} L_{t+1}, grad\_\widehat{\boldsymbol{h}}_{t+1})$
        if $t - 1 \in violations$ then
            $grad\_\boldsymbol{h}_t \leftarrow grad\_\boldsymbol{h}_t + grad\_\widehat{\boldsymbol{h}}_t$
            $grad\_\widehat{\boldsymbol{h}}_t \leftarrow \mathbf{0}$
```

prefix and the violating prefix, we end up with the relatively simple computation tree shown at the bottom of Figure 4.1. It is evident that we can backpropagate in a single pass, accumulating gradients from sequences that diverge from the gold at the time-step that precedes their divergence. The second half of Algorithm 5 shows this explicitly for a single sequence, though it is straightforward to extend the algorithm to operate in batch.[4]

[4] We also note that because we do not update the parameters until after the T'th search step, our training procedure differs slightly from LaSO (which is online), and in this aspect is essentially equivalent to the "delayed LaSO update" of Björkelund and Kuhn (2014).

4.4 Data and Methods

We run experiments on three different tasks, comparing our approach to the seq2seq baseline, and to other relevant baselines.

4.4.1 Model

While the method we describe applies to seq2seq RNNs in general, for all experiments we use the global attention model of Luong et al. (2015) — which consists of an LSTM (Hochreiter and Schmidhuber, 1997) encoder and an LSTM decoder with a global attention model — as both the baseline seq2seq model (i.e., as the model that computes the g in Equation (4.1)) and as the model that computes our sequence-scores $f(w_t, \boldsymbol{h}_{t-1}, \boldsymbol{x})$. As in Luong et al. (2015), we also use "input feeding," which involves feeding the attention distribution from the previous time-step into the decoder at the current step. This model architecture has been found to be highly performant for neural machine translation and other seq2seq tasks.

To distinguish the models we refer to our system as BSO (beam search optimization) and to the baseline as seq2seq. When we apply constrained training (as discussed in Section 4.3.2), we refer to the model as ConBSO. In providing results we also distinguish between the beam size K_{tr} with which the model is trained, and the beam size K_{te} which is used at test-time. In general, if we plan on evaluating with a beam of size K_{te} it makes sense to train with a beam of size $K_{tr} = K_{te} + 1$, since our objective requires the gold sequence to be scored higher than the *last* sequence on the beam.

4.4.2 Methodology

Here we detail additional techniques we found necessary to ensure the model learned effectively. First, we found that the model failed to learn when trained from a random initialization.[5] We therefore found it necessary to pre-train the model using a standard, word-level cross-entropy loss as in Equation (4.2). The necessity of pre-training in this instance is consistent with the findings of other authors who train non-local neural models (Kingsbury, 2009; Sak et al., 2014; Ranzato et al., 2016), though it is worth noting that Andor et al. (2016) found that pre-training was only necessary for increasing convergence-speed.

Similarly, it is clear that the smaller the beam used in training is, the less room the model has to make erroneous predictions without running afoul of the margin loss. Accordingly, we also found it useful to use a "curriculum beam" strategy in training, whereby the size of the beam is increased gradually during training. In particular, given a desired training beam size K_{tr}, we began training with a beam of size 2, and increased it by 1 every 2 epochs until reaching K_{tr}.

Finally, it has been established that *dropout* (Srivastava et al., 2014) regularization improves the performance of LSTMs (Pham et al., 2014; Zaremba et al., 2014), and in our experiments we run beam search under dropout. (However, it is important to ensure that the same mask applied at each time-step of the forward search is also applied at the corresponding step of the backward pass. We accomplish this by pre-computing masks for each time-step, and sharing them between the partial sequence LSTMs.) For all experiments, we trained both seq2seq and BSO models with mini-batch Adagrad (Duchi et al., 2011) (using batches

[5] This may be because there is relatively little signal in the sparse, sequence-level gradient, but this point requires further investigation.

of size 64), and we renormalized all gradients so they did not exceed 5 before updating parameters. We did not extensively tune learning-rates, but we found initial rates of 0.02 for the encoder and decoder LSTMs, and a rate of 0.1 or 0.2 for the final linear layer (i.e., the layer tasked with making word-predictions at each time-step) to work well across all the tasks we considered. Code implementing the experiments described below can be found at https://github.com/harvardnlp/BSO.[6]

4.4.3 Tasks and Results

Our experiments are primarily intended to evaluate the effectiveness of beam search optimization over standard seq2seq training. As such, we run experiments with the same model across three very different problems: word ordering, dependency parsing, and machine translation. While we do not include all the features and extensions necessary to reach state-of-the-art performance, even the baseline seq2seq model is generally quite performant.

Word Ordering The task of correctly ordering the words in a shuffled sentence has recently gained some attention as a way to test the (syntactic) capabilities of text-generation systems (Zhang and Clark, 2011, 2015; Liu et al., 2015; Schmaltz et al., 2016). We cast this task as seq2seq problem by viewing a shuffled sentence as a source sentence, and the correctly ordered sentence as the target. While word ordering is a somewhat synthetic task, it has two interesting properties for our purposes. First, it is a task which plausibly requires search (due to the exponentially many possible orderings), and, second, there is a clear hard constraint on output sequences, namely, that they be a permutation of the source sequence. For both the

[6] Our code is based on Yoon Kim's seq2seq code, https://github.com/harvardnlp/seq2seq-attn.

	Word Ordering (BLEU)		
	$K_{te} = 1$	$K_{te} = 5$	$K_{te} = 10$
seq2seq	25.2	29.8	31.0
BSO	28.0	33.2	34.3
ConBSO	**28.6**	**34.3**	**34.5**
LSTM-LM	15.4	-	26.8

Table 4.1: Word ordering. BLEU Scores of seq2seq, BSO, constrained BSO, and a vanilla LSTM language model (from Schmaltz et al, 2016). All experiments above have $K_{tr} = 6$.

baseline and BSO models we enforce this constraint at test-time. However, we also experiment with constraining the BSO model during training, as described in Section 4.3.2, by defining the succ function to only allow successor sequences containing un-used words in the source sentence.

For experiments, we use the same PTB dataset (with the standard training, development, and test splits) and evaluation procedure as in Zhang and Clark (2015) and later work, with performance reported in terms of BLEU score with the correctly ordered sentences. For all word-ordering experiments we use 2-layer encoder and decoder LSTMs, each with 256 hidden units, and dropout with a rate of 0.2 between LSTM layers. We use simple 0/1 costs in defining the Δ function. We show our test-set results in Table 4.1. We see that on this task there is a large improvement at each beam size from switching to BSO, and a further improvement from using the constrained model.

Inspired by a similar analysis in Daumé III and Marcu (2005), we further examine the relationship between K_{tr} and K_{te} when training with ConBSO in Table 4.2. We see that larger K_{tr} hurt greedy inference, but that results continue to improve, at least initially, when using a K_{te} that is (somewhat) bigger than $K_{tr} - 1$.

	Word Ordering Beam Size (BLEU)		
	$K_{te} = 1$	$K_{te} = 5$	$K_{te} = 10$
$K_{tr} = 2$	30.59	31.23	30.26
$K_{tr} = 6$	28.20	34.22	34.67
$K_{tr} = 11$	26.88	34.42	34.88
seq2seq	26.11	30.20	31.04

Table 4.2: Beam-size experiments on word ordering development set. All numbers reflect training with constraints (ConBSO).

Dependency Parsing We next apply our model to dependency parsing, which also has hard constraints and plausibly benefits from search. We treat dependency parsing with arc-standard transitions as a seq2seq task by attempting to map from a source sentence to a target sequence of source sentence words interleaved with the arc-standard, reduce-actions in its parse. For example, we attempt to map the source sentence

> But it was the Quotron problems that ...

to the target sequence

> But it was @L_SBJ @L_DEP the Quotron problems @L_NMOD @L_NMOD that ...

We use the standard Penn Treebank dataset splits with Stanford dependency labels, and the standard UAS/LAS evaluation metric (excluding punctuation) following Chen and Manning (2014). All models thus see only the words in the source and, when decoding, the actions it has emitted so far; no other features are used. We use 2-layer encoder and decoder LSTMs with 300 hidden units per layer and dropout with a rate of 0.3 between LSTM layers. We replace singleton words in the training set with an UNK token, normalize digits to a single symbol, and initialize word embeddings for both source and target words from the publicly available `word2vec` (Mikolov et al., 2013) embeddings. We use simple 0/1 costs in defining the Δ function.

	Dependency Parsing (UAS/LAS)		
	$K_{te} = 1$	$K_{te} = 5$	$K_{te} = 10$
seq2seq	**87.33/82.26**	88.53/84.16	88.66/84.33
BSO	86.91/82.11	91.00/**87.18**	91.17/**87.41**
ConBSO	85.11/79.32	**91.25**/86.92	**91.57**/87.26
Andor	93.17/91.18	-	-

Table 4.3: Dependency parsing. UAS/LAS of seq2seq, BSO, ConBSO and baselines on PTB test set. Andor is the current state-of-the-art model for this data set (Andor et al. 2016), and we note that with a beam of size 32 they obtain 94.41/92.55. All experiments above have $K_{tr} = 6$.

As in the word-ordering case, we also experiment with modifying the succ function in order to train under hard constraints, namely, that the emitted target sequence be a valid parse. In particular, we constrain the output at each time-step to obey the stack constraint, and we ensure words in the source are emitted in order.

We show results on the test-set in Table 4.3. BSO and ConBSO both show significant improvements over seq2seq, with ConBSO improving most on UAS, and BSO improving most on LAS. We achieve a reasonable final score of 91.57 UAS, which lags behind the state-of-the-art, but is promising for a general-purpose, word-only model.

Translation We finally evaluate our model on a small machine translation dataset, which allows us to experiment with a cost function that is not 0/1, and to consider other baselines that attempt to mitigate exposure bias in the seq2seq setting. We use the dataset from the work of Ranzato et al. (2016), which uses data from the German-to-English portion of the IWSLT 2014 machine translation evaluation campaign (Cettolo et al., 2014). The data comes from translated TED talks, and the dataset contains roughly 153K training sentences, 7K development sentences, and 7K test sentences. We use the same preprocessing and dataset splits as Ranzato et al. (2016), and like them we also use a single-layer LSTM decoder with 256

	Machine Translation (BLEU)		
	$K_{te} = 1$	$K_{te} = 5$	$K_{te} = 10$
seq2seq	22.53	24.03	23.87
BSO, SB-Δ	**23.83**	**26.36**	**25.48**
XENT	17.74	20.10	20.28
DAD	20.12	22.25	22.40
MIXER	20.73	21.81	21.83

Table 4.4: Machine translation experiments on test set; results below middle line are from MIXER model of Ranzato et al. (2016). SB-Δ indicates sentence BLEU costs are used in defining Δ. XENT is similar to our seq2seq model but with a convolutional encoder and simpler attention. DAD trains seq2seq with scheduled sampling (Bengio et al., 2015). BSO, SB-Δ experiments above have $K_{tr} = 6$.

units. We also use dropout with a rate of 0.2 between each LSTM layer. We emphasize, however, that while our decoder LSTM is of the same size as that of Ranzato et al. (2016), our results are not directly comparable, because we use an LSTM encoder (rather than a convolutional encoder as they do), a slightly different attention mechanism, and input feeding (Luong et al., 2015).

For our main MT results, we set $\Delta(\hat{y}_{1:t}^{(k)})$ to $1 - \text{SB}(\hat{y}_{r+1:t}^{(K)}, y_{r+1:t})$, where r is the last margin violation and SB denotes smoothed, sentence-level BLEU (Chen and Cherry, 2014). This setting of Δ should act to penalize erroneous predictions with a relatively low sentence-level BLEU score more than those with a relatively high sentence-level BLEU score. In Table 4.4 we show our final results and those from Ranzato et al. (2016).[7] While we start with an improved baseline, we see similarly large increases in accuracy as those obtained by DAD and MIXER, in particular when $K_{te} > 1$.

We further investigate the utility of these sequence-level costs in Table 4.5, which compares using sentence-level BLEU costs in defining Δ with using 0/1 costs. We see that the more sophisticated sequence-level costs have a moderate effect on BLEU score.

[7] Some results from personal communication.

	Machine Translation (BLEU)		
	$K_{te}=1$	$K_{te}=5$	$K_{te}=10$
0/1-Δ	25.73	28.21	27.43
SB-Δ	25.99	28.45	27.58

Table 4.5: BLEU scores obtained on the machine translation development data when training with $\Delta(\hat{y}_{1:t}^{(k)}) = 1$ (top) and $\Delta(\hat{y}_{1:t}^{(k)}) = 1 - \mathrm{SB}(\hat{y}_{r+1:t}^{(K)}, y_{r+1:t})$ (bottom), and K_{tr} = 6.

Timing Given Algorithm 5, we would expect training time to increase linearly with the size of the beam. On the above MT task, our highly tuned seq2seq baseline processes an average of 13,038 tokens/second (including both source and target tokens) on a GTX 970 GPU. For beams of size $K_{tr} = 2$, 3, 4, 5, and 6, our implementation processes on average 1,985, 1,768, 1,709, 1,521, and 1,458 tokens/second, respectively. Thus, we appear to pay an initial constant factor of ≈ 3.3 due to the more complicated forward and backward passes, and then training scales with the size of the beam. Because we batch beam predictions on a GPU, however, we find that in practice training time scales sub-linearly with the beam-size.

4.5 Recent Related Work

In this section we highlight recent related work not included in the discussion of now-classic structured prediction techniques in Chapter 2. When it comes to training RNNs, SEARN/DAgger has been applied under the name "scheduled sampling" (Bengio et al., 2015), which involves training an RNN to generate the $t+1$'st token in a target sequence after consuming either the true t'th token, or, with probability that increases throughout training, the predicted t'th token.

In the context of feed-forward neural network training, early update training has been recently explored in a feed-forward setting by Zhou et al. (2015) and Andor et al. (2016). Our work differs in that we adopt a LaSO-like paradigm (with some minor modifications), and apply it to the training of seq2seq RNNs (rather than feed-forward networks). We also note that Watanabe and Sumita (2015) apply maximum-violation training (Huang et al., 2012), which is similar to early-update, to a parsing model with recurrent components, and that Yazdani and Henderson (2015) use beam-search in training a discriminative, locally normalized dependency parser with recurrent components.

Recently authors have also proposed alleviating exposure bias using techniques from reinforcement learning. Ranzato et al. (2016) follow this approach to train RNN decoders in a seq2seq model, and they obtain consistent improvements in performance, even over models trained with scheduled sampling. As Daumé III and Marcu (2005) note, LaSO is similar to reinforcement learning, except it does not require "exploration" in the same way. Such exploration may be unnecessary in supervised text-generation, since we typically know the gold partial sequences at each time-step. Shen et al. (2016) use minimum risk training (approximated by sampling) to address the issues of exposure bias and loss-evaluation mismatch for seq2seq MT, and show impressive performance gains.

Whereas exposure bias results from training in a certain way, label bias results from properties of the model itself. In particular, label bias is likely to affect structured models that make sub-structure predictions using locally-normalized scores. Because the neural and non-neural literature on this point has recently been reviewed by Andor et al. (2016), we simply note here that RNN models are typically locally normalized, and we are unaware of any specifically seq2seq work with RNNs that does *not* use locally-normalized scores. The model we introduce here, however, is not locally normalized, and so should not suffer from label bias. We also note that there are some (non-seq2seq) exceptions to the trend of locally normalized

RNNs, such as the work of Sak et al. (2014) and Voigtlaender et al. (2015), who train LSTMs in the context of HMMs for speech recognition using sequence-level objectives; their work does not consider search, however.

4.6 Conclusion

We have introduced a variant of seq2seq and an associated beam search training scheme, which addresses exposure bias as well as label bias, and moreover allows for both training with sequence-level cost functions as well as with hard constraints. The strength of this approach is that it can be applied to any problem that might be formulated as a sequence transduction problem. At the same time, there are particular problems facing the important structured prediction problem of text generation that are *not* explicitly addressed by the BSO framework. We identify some of these problems in the next chapter.

5

Database-to-Document Generation[1]

5.1 Introduction

In this chapter we consider the structured prediction problem of text generation. While standard techniques for neural, structured prediction have shown tremendous success in generating relatively short translations or summaries, generating longer text remains a challenge. Indeed, as neural systems begin to move toward generating longer outputs in response to longer and more complicated inputs, the generated texts begin to display reference errors, inter-sentence

[1] Much of the the material in this chapter is adapted from Wiseman et al. (2017).

incoherence, and a lack of fidelity to the source material.

More fundamentally, there is at least one significant challenge associated with viewing text generation as a structured prediction problem in the same vein as coreference resolution or parsing, namely, that the conditional distribution we are attempting to model is extremely multimodal. For example, if we consider generating summaries y of some information x, the true distribution $p(y \,|\, x)$ will be extremely multimodal, simply because there are many correct ways to summarize information in natural language. Note that this differs from problems like coreference resolution, word-ordering, and parsing, where it is reasonable to assume there is a single correct structure for an input.

This multi-modality of the distribution we wish to model leads to two related problems. First, all of the losses we have been considering assume there is a single correct output $y^{(n)}$ for each input $x^{(n)}$; that is, they assume we are trying to model a unimodal distribution.[2] Second, most text generation datasets come only with a single reference generation per example, and so automatic evaluation inevitably compares the text generated by a system for each example to the single provided reference, making automatic evaluation of text generation systems notoriously problematic. The goal of this chapter is to suggest a particular, long-form generation task in which these challenges may be fruitfully explored, to provide a publicly available dataset for this task, to suggest some new automatic evaluation metrics, and finally to establish how current, neural text generation methods perform on this task.

A classic problem in natural-language generation (NLG) (Kukich, 1983; McKeown, 1992; Reiter and Dale, 1997) involves taking structured data, such as a table, as input, and producing text that adequately and fluently describes this data as output. Unlike machine transla-

[2] Note that locally normalized models can be interpreted as making a less strong assumption in this regard; they merely assume there is a single correct $y_t^{(n)}$ conditioned on x and $y_{1:t-1}^{(n)}$. This may partially explain why locally normalized RNNs as in the standard seq2seq setup are so effective.

tion, which aims for a complete transduction of the sentence to be translated, this form of NLG is typically taken to require addressing (at least) two separate challenges: *what to say*, the selection of an appropriate subset of the input data to discuss, and *how to say it*, the surface realization of a generation (Reiter and Dale, 1997; Jurafsky and Martin, 2014). Traditionally, these two challenges have been modularized and handled separately by generation systems. However, neural generation systems, which are typically trained end-to-end as conditional language models (Mikolov et al., 2010; Sutskever et al., 2011, 2014), blur this distinction.

In this context, we believe the problem of generating multi-sentence summaries of tables or database records to be a reasonable next-problem for neural techniques to tackle as they begin to consider more difficult NLG tasks. In particular, we would like this generation task to have the following two properties: (1) it is relatively easy to obtain fairly clean summaries and their corresponding databases for dataset construction, and (2) the summaries should be primarily focused on conveying the information in the database. This latter property ensures that the task is somewhat congenial to a standard encoder-decoder approach, and, more importantly, that it is reasonable to *evaluate* generations in terms of their fidelity to the database.

One task that meets these criteria is that of generating summaries of sports games from associated box-score data, and there is indeed a long history of NLG work that generates sports game summaries (Robin, 1994; Tanaka-Ishii et al., 1998; Barzilay and Lapata, 2005). To this end, we make the following contributions:

- We introduce a new large-scale corpus consisting of textual descriptions of basketball games paired with extensive statistical tables. This dataset is sufficiently large that fully data-driven approaches might be sufficient.
- We introduce a series of extractive evaluation models to automatically evaluate output generation performance, exploiting the fact that post-hoc information extraction is sig-

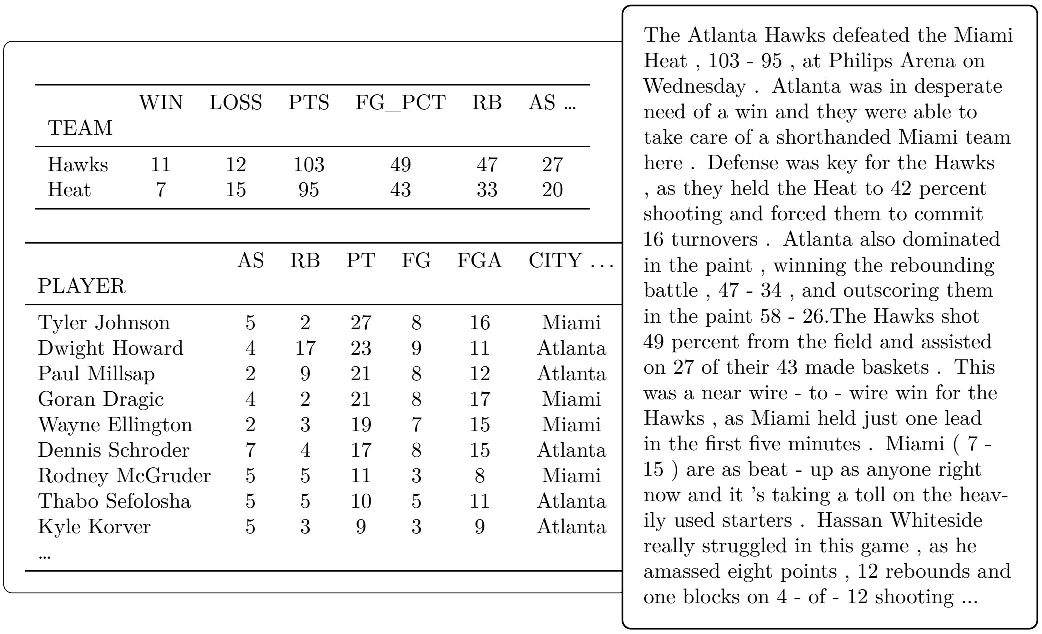
nificantly easier than generation itself.

- We apply a series of state-of-the-art neural methods, as well as a simple templated generation system, to our data-to-document generation task in order to establish baselines and study their generations.

Our experiments indicate that neural systems are quite good at producing fluent outputs and generally score well on standard word-match metrics, but perform quite poorly at content selection and at capturing long-term structure. While the use of copy-based models and additional reconstruction terms in the training loss can lead to improvements in BLEU and in our proposed extractive evaluations, current models are still quite far from producing human-level output, and are significantly worse than templated systems in terms of content selection and realization. Overall, we believe this problem of data-to-document generation highlights important remaining challenges in neural generation systems, and the use of extractive evaluation reveals significant issues hidden by standard automatic metrics.

5.2 Data-to-Text Datasets

We consider the problem of generating descriptive text from database records. Largely following the notation in Liang et al. (2009), let $x = \{r_j\}_{j=1}^{J}$ be a set of records, where for each $r \in x$ we define $r.t \in \mathcal{T}$ to be the *type* of r, and we assume each r to be a binarized relation, where $r.e$ and $r.m$ are a record's entity and value, respectively. For example, a database recording statistics for a basketball game might have a record r such that $r.t$ = points, $r.e$ = Russell Westbrook, and $r.m = 50$. In this case, $r.e$ gives the player in question, and $r.m$ gives the number of points the player scored. From these records, we are interested in generating descriptive text, $\hat{y}_{1:T} = \hat{y}_1, \ldots, \hat{y}_T$ of T words such that $\hat{y}_{1:T}$ is an adequate and fluent summary of x. A dataset for training data-to-document systems typically consists of $(x, y_{1:T})$ pairs,



TEAM	WIN	LOSS	PTS	FG_PCT	RB	AS ...
Hawks	11	12	103	49	47	27
Heat	7	15	95	43	33	20

PLAYER	AS	RB	PT	FG	FGA	CITY . . .
Tyler Johnson	5	2	27	8	16	Miami
Dwight Howard	4	17	23	9	11	Atlanta
Paul Millsap	2	9	21	8	12	Atlanta
Goran Dragic	4	2	21	8	17	Miami
Wayne Ellington	2	3	19	7	15	Miami
Dennis Schroder	7	4	17	8	15	Atlanta
Rodney McGruder	5	5	11	3	8	Miami
Thabo Sefolosha	5	5	10	5	11	Atlanta
Kyle Korver	5	3	9	3	9	Atlanta
...						

The Atlanta Hawks defeated the Miami Heat , 103 - 95 , at Philips Arena on Wednesday . Atlanta was in desperate need of a win and they were able to take care of a shorthanded Miami team here . Defense was key for the Hawks , as they held the Heat to 42 percent shooting and forced them to commit 16 turnovers . Atlanta also dominated in the paint , winning the rebounding battle , 47 - 34 , and outscoring them in the paint 58 - 26.The Hawks shot 49 percent from the field and assisted on 27 of their 43 made baskets . This was a near wire - to - wire win for the Hawks , as Miami held just one lead in the first five minutes . Miami (7 - 15) are as beat - up as anyone right now and it 's taking a toll on the heavily used starters . Hassan Whiteside really struggled in this game , as he amassed eight points , 12 rebounds and one blocks on 4 - of - 12 shooting ...

Figure 5.1: An example data-record and document pair from the RotoWire dataset. We show a subset of the game's records (there are 628 in total), and a selection from the gold document. The document mentions only a select subset of the records, but may express them in a complicated manner. In addition to capturing the writing style, a generation system should select similar record content, express it clearly, and order it appropriately.

where $y_{1:T}$ is a document consisting of a gold (i.e., human generated) summary for database x.

Several benchmark datasets have been used in recent years for the text generation task, the most popular of these being WeatherGov (Liang et al., 2009) and Robocup (Chen and Mooney, 2008). Recently, neural generation systems have show strong results on these datasets, with the system of Mei et al. (2016) achieving BLEU scores in the 60s and 70s on WeatherGov, and BLEU scores of almost 30 even on the smaller Robocup dataset. These results are quite promising, and suggest that neural models are a good fit for text generation. However, the statistics of these datasets, shown in Table 5.1, indicate that these datasets use relatively simple language and record structure. Furthermore, there is reason to believe that WeatherGov is at least partially machine-generated (Reiter, 2017). More recently, Lebret et al. (2016) introduced the WikiBio dataset, which is at least an order of magnitude larger in terms of number of tokens and record types. However, as shown in Table 5.1, this dataset too only contains short (single-sentence) generations, and relatively few records per generation. As such, we believe that early success on these datasets is not yet sufficient for testing the desired linguistic capabilities of text generation at a document-scale.

With this challenge in mind, we introduce a new dataset for data-to-document text generation, available at `https://github.com/harvardnlp/boxscore-data`. The dataset is intended to be comparable to WeatherGov in terms of token count, but to have significantly longer target texts, a larger vocabulary space, and to require more difficult content selection.

The dataset consists of two sources of articles summarizing NBA basketball games, paired with their corresponding box- and line-score tables. The data statistics of these two sources, RotoWire and SBNation, are also shown in Table 5.1. The first dataset, RotoWire, uses

	RC	WG	WB	RW	SBN
Vocab	409	394	400K	11.3K	68.6K
Tokens	11K	0.9M	19M	1.6M	8.8M
Examples	1.9K	22.1K	728K	4.9K	10.9K
Avg Len	5.7	28.7	26.1	337.1	805.4
Rec. Types	4	10	1.7K	39	39
Avg Records	2.2	191	19.7	628	628

Table 5.1: Vocabulary size, number of total tokens, number of distinct examples, average generation length, total number of record types, and average number of records per example for the Robocup (RC), WeatherGov (WG), WikiBio (WB), RotoWire (RW), and SBNation (SBN) datasets.

professionally written, medium length game summaries targeted at fantasy basketball fans. The writing is colloquial, but relatively well structured, and targets an audience primarily interested in game statistics. The second dataset, SBNation, uses fan-written summaries targeted at other fans. This dataset is significantly larger, but also much more challenging, as the language is very informal, and often tangential to the statistics themselves. We show some sample text from RotoWire in Figure 5.1. Our primary focus will be on the RotoWire data.

5.3 Evaluating Document Generation

We begin by discussing the evaluation of generated documents, since both the task we introduce and the evaluation methods we propose are motivated by some of the shortcomings of current approaches to evaluation. Text generation systems are typically evaluated using a combination of automatic measures, such as BLEU (Papineni et al., 2002), and human evaluation. While BLEU is perhaps a reasonably effective way of evaluating short-form text generation, we found it to be unsatisfactory for document generation. In particular, we note that it primarily rewards fluent text generation, rather than generations that capture the most important information in the database, or that report the information in a particularly coherent way. While human evaluation, on the other hand, is likely ultimately necessary for evaluating generations (Liu et al., 2016; Wu et al., 2016), it is much less convenient than using automatic metrics. Furthermore, we believe that current text generations are sufficiently bad in sufficiently obvious ways that automatic metrics can still be of use in evaluation, and we are not yet at the point of needing to rely solely on human evaluators.

5.3.1 Extractive Evaluation

To address this evaluation challenge, we begin with the intuition that assessing document quality is easier than document generation. In particular, it is much easier to automatically extract information from documents than to generate documents that accurately convey desired information. As such, simple, high-precision information extraction models can serve as the basis for assessing and better understanding the quality of automatic generations. We emphasize that such an evaluation scheme is most appropriate when evaluating generations (such as basketball game summaries) that are primarily intended to summarize information. While many generation problems do not fall into this category, we believe this to be an interesting category, and one worth focusing on *because* it is amenable to this sort of evaluation.

To see how a simple information extraction system might work, consider the document in Figure 5.1. We may first extract candidate entity (player, team, and city) and value (number and certain string) pairs $r.e, r.m$ that appear in the text, and then predict the type $r.t$ (or none) of each candidate pair. For example, we might extract the entity-value pair ("Miami Heat", "95") from the first sentence in Figure 5.1, and then predict that the *type* of this pair is POINTS, giving us an extracted record r such that $(r.e, r.m, r.t)$ = (MIAMI HEAT, 95, POINTS). Indeed, many relation extraction systems reduce relation extraction to multi-class classification precisely in this way (Zhang, 2004; Zhou et al., 2008; Zeng et al., 2014; dos Santos et al., 2015).

More concretely, given a document $\hat{y}_{1:T}$, we consider all pairs of word-spans in each sentence that represent possible entities e and values m. We then model $p(r.t \mid e, m; \theta)$ for each pair, using $r.t = \epsilon$ to indicate unrelated pairs. We use architectures similar to those discussed in Collobert et al. (2011) and dos Santos et al. (2015) to parameterize this probability; full details are given in Appendix A.

Importantly, we note that the $(x, y_{1:T})$ pairs typically used for training data-to-document systems are also sufficient for training the information extraction model presented above, since we can obtain (partial) supervision by simply checking whether a candidate record lexically matches a record in x.[3] However, since there may be multiple records $r \in x$ with the same e and m but with different types $r.t$, we will not always be able to determine the type of a given entity-value pair found in the text. We therefore train our classifier to minimize a latent-variable loss: for all document spans e and m, with observed types $t(e, m) = \{r.t : r \in x, r.e = e, r.m = m\}$ (possibly $\{\epsilon\}$), we minimize

$$L(\theta) = -\sum_{e,m} \log \sum_{t' \in t(e,m)} p(r.t = t' \mid e, m; \theta).$$

We find that this simple system trained in this way is quite accurate at predicting relations. On the ROTOWIRE data it achieves over 90% accuracy on held-out data, and recalls approximately 60% of the relations licensed by the records.

[3] Alternative approaches explicitly align the document with the table for this task (Liang et al., 2009).

5.3.2 Comparing Generations

With a sufficiently precise relation extraction system, we can begin to evaluate how well an automatic generation $\hat{y}_{1:T}$ has captured the information in a set of records x. In particular, since the predictions of a precise information extraction system serve to align entity-mention pairs in the text with database records, this alignment can be used both to evaluate a generation's content selection ("what the generation says"), as well as content placement ("how the generation says it").

We consider in particular three induced metrics:

- Content Selection (CS): precision and recall of unique relations r extracted from $\hat{y}_{1:T}$ that are also extracted from $y_{1:T}$. This measures how well the generated document matches the gold document in terms of selecting which records to generate.
- Relation Generation (RG): precision and number of unique relations r extracted from $\hat{y}_{1:T}$ that also appear in x. This measures how well the system is able to generate text containing factual (i.e., correct) records.
- Content Ordering (CO): normalized Damerau-Levenshtein Distance (Brill and Moore, 2000)[4] between the sequences of records extracted from $y_{1:T}$ and that extracted from $\hat{y}_{1:T}$. This measures how well the system orders the records it chooses to discuss.

We note that CS primarily targets the "what to say" aspect of evaluation, CO targets the "how to say it" aspect, and RG targets both.

We conclude this section by contrasting the automatic evaluation we have proposed with recently proposed *adversarial evaluation* approaches, which also advocate automatic metrics backed by classification (Bowman et al., 2016; Kannan and Vinyals, 2016; Li et al., 2017). Unlike adversarial evaluation, which uses a black-box classifier to determine the quality of a generation, our metrics are defined with respect to the predictions of an information extraction

[4] DLD is a variant of Levenshtein distance that allows transpositions of elements; it is useful in comparing the ordering of sequences that may not be permutations of the same set (which is a requirement for measures like Kendall's Tau).

system. Accordingly, our metrics are quite interpretable, since by construction it is always possible to determine which fact (i.e., entity-value pair) in the generation is determined by the extractor to not match the database or the gold generation.

5.4 Neural Data-to-Document Models

In this section we briefly describe the neural generation methods we apply to the proposed task. As a base model we utilize the now standard attention-based encoder-decoder model (Sutskever et al., 2014; Cho et al., 2014; Bahdanau et al., 2015). We also experiment with several recent extensions to this model, including copy-based generation, and training with a source reconstruction term in the loss (in addition to the standard per-target-word loss).

Base Model For our base model, we map each record $r \in x$ into a vector $\tilde{\boldsymbol{r}}$ by first embedding $r.t$ (e.g., points), $r.e$ (e.g., Russell Westbrook), and $r.m$ (e.g., 50), and then applying a 1-layer MLP (similar to Yang et al. (2016)).[5] Our source data-records are then represented as $\tilde{x} = \{\tilde{\boldsymbol{r}}_j\}_{j=1}^{J}$. Given $\tilde{x}$, we use an LSTM decoder with attention and input-feeding, in the style of Luong et al. (2015), to compute the probability of each target word, conditioned on the previous words and on x. The model is trained end-to-end to minimize the negative log-likelihood of the words in the gold text $y_{1:T}$ given corresponding source material x.

Copying There has been a surge of recent work involving augmenting encoder-decoder models to copy words directly from the source material on which they condition (Gu et al., 2016; Gülçehre et al., 2016; Merity et al., 2016; Jia and Liang, 2016; Yang et al., 2016). These mod-

[5] We also include an additional feature for whether the player is on the home- or away-team.

els typically introduce an additional binary variable z_t into the per-timestep target word distribution, which indicates whether the target word $\hat{y}_t$ is copied from the source or generated:

$$p(\hat{y}_t \,|\, \hat{y}_{1:t-1}, x) = \sum_{z \in \{0,1\}} p(\hat{y}_t, z_t = z \,|\, \hat{y}_{1:t-1}, x).$$

In our case, we assume that target words are copied from the *value* portion of a record r; that is, a copy implies $\hat{y}_t \!=\! r.m$ for some r and t.

Joint Copy Model The models of Gu et al. (2016) and Yang et al. (2016) parameterize the *joint* distribution table over $\hat{y}_t$ and z_t directly:

$$p(\hat{y}_t, z_t \,|\, \hat{y}_{1:t-1}, x) \propto \begin{cases} \text{copy}(\hat{y}_t, \hat{y}_{1:t-1}, x) & z_t = 1,\ \hat{y}_t \in x \\ 0 & z_t = 1,\ \hat{y}_t \not\in x \\ \text{gen}(\hat{y}_t, \hat{y}_{1:t-1}, x) & z_t = 0, \end{cases}$$

where copy and gen are functions parameterized in terms of the decoder RNN's hidden state that assign scores to words, and where the notation $\hat{y}_t \!\in\! x$ indicates that $\hat{y}_t$ is equal to $r.m$ for some $r \!\in\! x$.

Conditional Copy Model Gülçehre et al. (2016), on the other hand, decompose the joint probability as:

$$p(\hat{y}_t, z_t \,|\, \hat{y}_{1:t-1}, x) = \begin{cases} p_{\text{copy}}(\hat{y}_t \,|\, z_t, \hat{y}_{1:t-1}, x)\, p(z_t \,|\, \hat{y}_{1:t-1}, x) & z_t{=}1 \\ p_{\text{gen}}(\hat{y}_t \,|\, z_t, \hat{y}_{1:t-1}, x)\, p(z_t \,|\, \hat{y}_{1:t-1}, x) & z_t{=}0, \end{cases}$$

where an MLP is used to model $p(z_t \,|\, \hat{y}_{1:t-1}, x)$.

Models with copy-decoders may be trained to minimize the negative log marginal probability, marginalizing out the latent-variable z_t (Gu et al., 2016; Yang et al., 2016; Merity et al., 2016). However, if it is known which target words y_t are copied, it is possible to train with a loss that does not marginalize out the latent z_t. Gülçehre et al. (2016), for instance, assume that any target word y_t that also appears in the source is copied, and train to minimize the negative joint log-likelihood of the y_t and z_t.

In applying such a loss in our case, we again note that there may be multiple records r such that $r.m$ appears in $\hat{y}_{1:T}$. Accordingly, we slightly modify the p_{copy} portion of the loss of Gülçehre et al. (2016) to sum over all matched records. In particular, we model the probability of relations $r \in x$ such that $r.m = y_t$ and $r.e$ is in the same sentence as $r.m$. Letting $r(y_t) = \{r \in x : r.m = y_t, \text{same}{-}\text{sentence}(r.e, r.m)\}$, we have:

$$p_{\text{copy}}(y_t \,|\, z_t, y_{1:t-1}, x) = \sum_{r \in r(y_t)} p(r \,|\, z_t, y_{1:t-1}, x).$$

We note here that the key distinction for our purposes between the Joint Copy model and the Conditional Copy model is that the latter *conditions* on whether there is a copy or not, and so in p_{copy} the source records compete only with each other. In the Joint Copy model, however, the source records also compete with words that cannot be copied. As a result, training the Conditional Copy model with the supervised loss of Gülçehre et al. (2016) can be seen as training with a word-level reconstruction loss, where the decoder is trained to choose the record in x that gives rise to y_t.

Reconstruction Losses Reconstruction-based techniques can also be applied at the document- or sentence-level during training. One simple approach to this problem is to utilize the hidden states of the decoder to try to reconstruct the database. A fully differentiable approach using the decoder hidden states has recently been successfully applied to neural machine translation by Tu et al. (2017). Unlike copying, this method is applied only at training, and attempts to learn decoder hidden states with broader coverage of the input data.

In adopting this reconstruction approach we segment the decoder hidden states $\boldsymbol{h}_t$ into $\lceil \frac{T}{B} \rceil$ contiguous blocks of size at most B. Denoting a single one of these hidden state blocks as $\boldsymbol{b}_i$, we attempt to predict each field value in some record $r \in x$ from $\boldsymbol{b}_i$. We define $p(r.e, r.m \,|\, \boldsymbol{b}_i)$, the probability of the entity and value in record r given $\boldsymbol{b}_i$, to be $\mathrm{softmax}(f(\boldsymbol{b}_i))$, where f is a parameterized function of $\boldsymbol{b}_i$, which in our experiments utilize a convolutional layer followed by an MLP; full details are given in Appendix A. We further extend this idea and predict K records in x from $\boldsymbol{b}_i$, rather than one. We can train with the following reconstruction loss for a particular $\boldsymbol{b}_i$:

$$L(\theta) = -\sum_{k=1}^{K} \min_{r \in x} \log p_k(r \,|\, \boldsymbol{b}_i; \theta) = -\sum_{k=1}^{K} \min_{r \in x} \sum_{x \in \{e,m,t\}} \log p_k(r.x \,|\, \boldsymbol{b}_i; \theta),$$

where p_k is the k'th predicted distribution over records, and where we have modeled each component of r independently. This loss attempts to make the *most* probable record in x given $\boldsymbol{b}_i$ more probable. We found that augmenting the above loss with a term that penalizes the total variation distance (TV) between the p_k to be helpful.[6] Both $L(\theta)$ and the TV distance term

[6] Penalizing the TV distance between the p_k might be useful if, for instance, K is too large, and only a smaller number of records can be predicted from $\boldsymbol{b}_i$. We also experimented with *encouraging*, rather than penalizing the TV distance between the p_k, which might make sense if we were worried about ensuring the p_k captured different records.

are simply added to the standard negative log-likelihood objective at training time.

5.5 Experimental Methods

In this section we highlight a few important details of our models and methods; full details are in Appendix A. For our RotoWire models, the record encoder produces $\tilde{\boldsymbol{r}}_j$ in $\mathbb{R}^{600}$, and we use a 2-layer LSTM decoder with hidden states of the same size as the $\tilde{\boldsymbol{r}}_j$, and dot-product attention and input-feeding in the style of Luong et al. (2015). Unlike past work, we use two identically structured attention layers, one to compute the standard generation probabilities (gen or p_{gen}), and one to produce the scores used in copy or p_{copy}.

We train the generation models using SGD and truncated BPTT (Elman, 1990; Mikolov et al., 2010), as in language modeling. That is, we split each $y_{1:T}$ into contiguous blocks of length 100, and backprop both the gradients with respect to the current block as well as with respect to the encoder parameters for each block.

Our extractive evaluator consists of an ensemble of 3 single-layer convolutional and 3 single-layer bidirectional LSTM models. The convolutional models concatenate convolutions with kernel widths 2, 3, and 5, and 200 feature maps in the style of (Kim, 2014). Both models are trained with SGD.

Templatized Generator In addition to neural baselines, we also use a problem-specific, template-based generator. The template-based generator first emits a sentence about the teams playing in the game, using a templatized sentence taken from the training set:

> The <team1> (<wins1>-<losses1>) defeated the <team2> (<wins2>-<losses2>) <pts1>-<pts2>.

Then, 6 player-specific sentences of the following form are emitted (again adapting a simple sentence from the training set):

> `<player>` scored `<pts>` points (`<fgm>`-`<fga>` FG, `<tpm>`-`<tpa>` 3PT, `<ftm>`-`<fta>` FT) to go with `<reb>` rebounds.

The 6 highest-scoring players in the game are used to fill in the above template. Finally, a typical end sentence is emitted:

> The `<team1>`' next game will be at home against the Dallas Mavericks, while the `<team2>` will travel to play the Bulls.

Code implementing all models can be found at https://github.com/harvardnlp/data2text. Our encoder-decoder models are based on OpenNMT (Klein et al., 2017).

5.6 Results

We found that all models performed quite poorly on the SBNation data, with the best model achieving a validation perplexity of 33.34 and a BLEU score of 1.78. This poor performance is presumably attributable to the noisy quality of the SBNation data, and the fact that many documents in the dataset focus on information not in the box- and line-scores. Accordingly, we focus on RotoWire in what follows.

The main results for the RotoWire dataset are shown in Table 5.2, which shows the performance of the models in Section 5.4 in terms of the metrics defined in Section 5.3.2, as well as in terms of perplexity and BLEU.

5.6.1 Discussion

There are several interesting relationships in the development portion of Table 5.2. First we note that the Template model scores very poorly on BLEU, but does quite well on the extrac-

		Development						
		RG		CS		CO	PPL	BLEU
Beam	Model	P%	#	P%	R%	DLD%		
	Gold	91.77	12.84	100	100	100	1.00	100
	Template	99.35	49.7	18.28	65.52	12.2	N/A	6.87
1	Joint Copy	47.55	7.53	20.53	22.49	8.28	7.46	10.41
	Joint Copy + Rec	57.81	8.31	23.65	23.30	9.02	7.25	10.00
	Joint Copy + Rec + TV	60.69	8.95	23.63	24.10	8.84	7.22	12.78
	Conditional Copy	68.94	9.09	25.15	22.94	9.00	7.44	13.31
5	Joint Copy	47.00	10.67	16.52	26.08	7.28	7.46	10.23
	Joint Copy + Rec	62.11	10.90	21.36	26.26	9.07	7.25	10.85
	Joint Copy + Rec + TV	57.51	11.41	18.28	25.27	8.05	7.22	12.04
	Conditional Copy	71.07	12.61	21.90	27.27	8.70	7.44	14.46
		Test						
	Template	99.30	49.61	18.50	64.70	8.04	N/A	6.78
5	Joint Copy + Rec	61.23	11.02	21.56	26.45	9.06	7.47	10.88
1	Joint Copy + Rec + TV	60.27	9.18	23.11	23.69	8.48	7.42	12.96
5	Conditional Copy	71.82	12.82	22.17	27.16	8.68	7.67	14.49

Table 5.2: Performance of induced metrics on gold and system outputs of RotoWire development and test data. Columns indicate Record Generation (RG) precision and count, Content Selection (CS) precision and recall, Count Ordering (CO) in normalized Damerau-Levenshtein distance, perplexity, and BLEU. These first three metrics are described in Section 5.3.2. Models compare Joint and Conditional Copy also with addition Reconstruction loss and Total Variation Distance extensions (described in Section 5.4).

tive metrics, providing an upper-bound for how domain knowledge could help content selection and generation. All the neural models make significant improvements in terms of BLEU score, with the conditional copying with beam search performing the best, even though all the neural models achieve roughly the same perplexity.

The extractive metrics provide further insight into the behavior of the models. We first note that on the gold documents $y_{1:T}$, the extractive model reaches 92% precision. Using the Joint Copy model, generation only has a record generation (RG) precision of 47% indicating that relationships are often generated incorrectly. The best Conditional Copy system improves this value to 71%, a significant improvement and potentially the cause of the improved BLEU score, but still far below gold.

Notably, content selection (CS) and content ordering (CO) seem to have no correlation at all with BLEU. There is some improvement with CS for the conditional model or reconstruction loss, but not much change as we move to beam search. CO actually gets worse as beam search is utilized, possibly a side effect of generating more records (RG#). The fact that these scores are much worse than the simple templated model indicates that further research is needed into better copying alone for content selection and better long term content ordering models.

Test results are consistent with development results, indicating that the Conditional Copy model is most effective at BLEU, RG, and CS, and that reconstruction is quite helpful for improving the joint model.

5.6.2 Human Evaluation

We also undertook two human evaluation studies, using Amazon Mechanical Turk. The first study attempted to determine whether generations considered to be more precise by our metrics were also considered more precise by human raters. To accomplish this, raters were presented with a particular NBA game's box score and line score, as well as with (randomly selected) sentences from summaries generated by our different models for those games. Raters were then asked to count how many facts in each sentence were supported by records in the box or line scores, and how many were contradicted. We randomly selected 20 distinct games to present to raters, and a total of 20 generated sentences per game were evaluated by raters. The left two columns of Table 5.3 contain the average numbers of supporting and contradicting facts per sentence as determined by the raters, for each model. We see that these results are generally in line with the RG and CS metrics, with the Conditional Copy model having the highest number of supporting facts, and the reconstruction terms significantly improving the Joint Copy models.

Using a Tukey HSD post-hoc analysis of an ANOVA with the number of contradicting facts as the dependent variable and the generating model and rater id as independent variables, we found significant ($p < 0.01$) pairwise differences in contradictory facts between the gold generations and all models except "Copy+Rec+TV," as well as a significant difference between "Copy+Rec+TV" and "Copy". We similarly found a significant pairwise difference between "Copy+Rec+TV" and "Copy" for number of supporting facts.

Our second study attempted to determine whether generated summaries differed in terms of how natural their *ordering* of records (as captured, for instance, by the DLD metric) is. To test this, we presented raters with random summaries generated by our models and asked

	# Supp.	# Cont.	Order Rat.
Gold	2.04	0.70	5.19
Joint Copy	1.65	2.31	3.90
Joint Copy + Rec	2.33	1.83	4.43
Joint Copy + Rec +TV	2.43	1.16	4.18
Conditional Copy	3.05	1.48	4.03

Table 5.3: Average rater judgment of number of box score fields supporting (left column) or contradicting (middle column) a generated sentence, and average rater Likert rating for the naturalness of a summary's ordering (right column). All generations use B=1.

them to rate the naturalness of the ordering of facts in the summaries on a 1-7 Likert scale. 30 random summaries were used in this experiment, each rated 3 times by distinct raters. The average Likert ratings are shown in the rightmost column of Table 5.3. While it is encouraging that the gold summaries received a higher average score than the generated summaries (and that the reconstruction term again improved the Joint Copy model), a Tukey HSD analysis similar to the one presented above revealed no significant pairwise differences.

5.6.3 Results on a Cleaner Dataset

Subsequent to the publication of the results in Table 5.2, another researcher discovered a few hundred examples where the team names and cities were attached to the wrong rows in the line-score portion of the data; that is, the home team name and city was associated with the away team's team-level statistics, and vice versa. The player-level information in the box score was unaffected. We present the results of retraining the best performing models from Table 5.2 in Table 5.4. There, we see that the major trends identified above still hold, with the templated model still outperforming the neural models by a wide margin on most metrics, though the neural models perform a few points higher on the induced metrics, as we might expect.

					Development			
		RG		CS		CO	PPL	BLEU
Beam	Model	P%	#	P%	R%	DLD%		
	Gold	95.98	16.93	100	100	100	1.00	100
	Template	99.93	54.21	23.42	72.62	11.30	N/A	8.97
1	Joint Copy + Rec + TV	61.23	15.27	28.79	39.80	15.27	7.26	12.69
	Conditional Copy	76.66	12.88	37.98	35.46	16.70	7.29	13.60
5	Joint Copy + Rec + TV	62.84	16.77	27.23	40.60	14.47	7.26	13.44
	Conditional Copy	75.74	16.93	31.20	38.94	14.98	7.29	14.57
					Test			
	Gold	96.11	17.31	100	100	100	1	100
	Template	99.95	54.15	23.74	72.36	11.68	N/A	8.93
5	Joint Copy + Rec + TV	62.66	16.82	27.60	40.59	14.57	7.49	13.61
5	Conditional Copy	75.62	16.83	32.80	39.93	15.62	7.53	14.19

Table 5.4: Performance of induced metrics on gold and system outputs of cleaned RotoWire development and test data. Columns indicate Record Generation (RG) precision and count, Content Selection (CS) precision and recall, Count Ordering (CO) in normalized Damerau-Levenshtein distance, perplexity, and BLEU. These first three metrics are described in Section 5.3.2. Models compare Joint and Conditional Copy also with addition Reconstruction loss and Total Variation Distance extensions (described in Section 5.4).

5.6.4 Qualitative Example

Figure 5.2 shows a document generated by the Conditional Copy model, using a beam of size 5. This particular generation evidently has several nice properties: it nicely learns the colloquial style of the text, correctly using idioms such as “19 percent from deep.” It is also partially accurate in its use of the records; we highlight in blue when it generates text that is licensed by a record in the associated box- and line-scores.

At the same time, the generation also contains major logical errors. First, there are basic copying mistakes, such as flipping the teams’ win/loss records. The system also makes obvious semantic errors; for instance, it generates the phrase “the Rockets were able to out-rebound the Rockets.” Finally, we see the model hallucinates factual statements, such as “in front of their home crowd,” which is presumably likely according to the language model, but ulti-

> The Utah Jazz (38 - 26) defeated the Houston Rockets (38 - 26) 117 - 91 on Wednesday at Energy Solutions Arena in Salt Lake City . The Jazz got out to a quick start in this one , out - scoring the Rockets 31 - 15 in the first quarter alone . Along with the quick start , the Rockets were the superior shooters in this game , going 54 percent from the field and 43 percent from the three - point line , while the Jazz went 38 percent from the floor and a meager 19 percent from deep . The Rockets were able to out - rebound the Rockets 49 - 49 , giving them just enough of an advantage to secure the victory in front of their home crowd . The Jazz were led by the duo of Derrick Favors and James Harden . Favors went 2 - for - 6 from the field and 0 - for - 1 from the three - point line to score a game - high of 15 points , while also adding four rebounds and four assists ....

Figure 5.2: Example document generated by the Conditional Copy system with a beam of size 5. Text that accurately reflects a record in the associated box- or line-score is highlighted in blue, and erroneous text is highlighted in red.

mately incorrect (and not supported by anything in the box- or line- scores). In practice, our proposed extractive evaluation will pick up on many errors in this passage. For instance, "four assists" is an RG error, repeating the Rockets' rebounds could manifest in a lower CO score, and incorrectly indicating the win/loss records is a CS error.

5.7 Related Work

In this section we note additional related work not noted throughout. Natural language generation has been studied for decades (Kukich, 1983; McKeown, 1992; Reiter and Dale, 1997), and generating summaries of sports games has been a topic of interest for almost as long (Robin, 1994; Tanaka-Ishii et al., 1998; Barzilay and Lapata, 2005).

Historically, research has focused on both content selection ("what to say") (Kukich, 1983; McKeown, 1992; Reiter and Dale, 1997; Duboue and McKeown, 2003; Barzilay and Lapata, 2005), and surface realization ("how to say it") (Goldberg et al., 1994; Reiter et al., 2005) with earlier work using (hand-built) grammars, and later work using SMT-like approaches (Wong and Mooney, 2007) or generating from PCFGs (Belz, 2008) or other formalisms (Soricut and Marcu, 2006; White et al., 2007). In the late 2000s and early 2010s, a number of systems were proposed that did both (Liang et al., 2009; Angeli et al., 2010; Kim and Mooney, 2010; Lu and Ng, 2011; Konstas and Lapata, 2013).

Within the world of neural text generation, some recent work has focused on conditioning language models on tables (Yang et al., 2016), and generating short biographies from Wikipedia Tables (Lebret et al., 2016; Chisholm et al., 2017). Mei et al. (2016) use a neural encoder-decoder approach on standard record-based generation datasets, obtaining impressive results, and motivating the need for more challenging NLG problems.

5.8 Conclusion and Future Work

This work explores the challenges facing neural data-to-document generation by introducing a new dataset, and proposing various metrics for automatically evaluating content selection, generation, and ordering. We see that recent ideas in copying and reconstruction lead to improvements on this task, but that there is a significant gap even between these neural models and templated systems. We hope to motivate researchers to focus further on generation problems that are relevant both to content selection and surface realization, but may not be reflected clearly in the model's perplexity.

Future work on this task might include approaches that process or attend to the source records in a more sophisticated way, generation models that attempt to incorporate semantic or reference-related constraints, and approaches to conditioning on facts or records that are not as explicit in the box- and line-scores.

A

Appendix to Chapter 5

A.1 Additional Dataset Details

The RotoWire data covers NBA games played between 1/1/2014 and 3/29/2017; some games have multiple summaries. The summaries have been randomly split into training, validation, and test sets consisting of 3398, 727, and 728 summaries, respectively.

The SBNation data covers NBA games played between 11/3/2006 and 3/26/2017; some games have multiple summaries. The summaries have been randomly split into training, validation, and test sets consisting of 7633, 1635, and 1635 summaries, respectively.

Player Types	POSN	MIN	PTS	FGM	FGA	FG-PCT	FG3M	FG3A	FG3-PCT
	FTM	FTA	FT-PCT	OREB	DREB	REB	AST	TOV	STL
	BLK	PF	FULL-NAME	NAME1	NAME2	CITY			
Team Types	PTS-QTR1	PTS-QTR2	PTS-QTR3	PTS-QTR4	PTS	FG-PCT	FG3-PCT	FT-PCT	REB
	AST	TOV	WINS	LOSSES	CITY	NAME			

Table A.1: Possible Record Types

All numbers in the box- and line-scores (but not the summaries) are converted to integers; fractional numbers corresponding to percents are multiplied by 100 to obtain integers in $[0, 100]$. We show the *types* of records in the data in Table A.1.

A.2 Generation Model Details

Encoder For the RotoWire data, a relation r is encoded into $\tilde{\boldsymbol{r}}$ by embedding each of $r.e$, $r.t$, $r.m$ and a "home-or-away" indicator feature in $\mathbb{R}^{600}$, and applying a 1-layer MLP (with ReLU nonlinearity) to map the concatenation of these vectors back into $\mathbb{R}^{600}$. To initialize the decoder LSTMs, we first mean-pool over the $\tilde{\boldsymbol{r}}_j$ by entity (giving one vector per entity), and then linearly transform the concatenation of these pooled entity-representations so that they can initialize the cells and hidden states of a 2-layer LSTM with states also in $\mathbb{R}^{600}$. The SBNation setup is identical, except all vectors are in $\mathbb{R}^{700}$.

Decoder As mentioned in the body of the paper, we compute two different attention distributions (i.e., using different parameters) at each decoding step. For the Joint Copy model, one attention distribution is not normalized, and is normalized along with all the output-word probabilities.

Within the Conditional Copy model we compute $p(z_t|\hat{y}_{1:t-1}, \boldsymbol{s})$ by mean-pooling the $\tilde{\boldsymbol{r}}_j$, concatenating them with the current (topmost) hidden state of the LSTM, and then feeding this concatenation via a 1-layer ReLU MLP with hidden dimension 600, and with a Sigmoid output layer.

For the reconstruction-loss, we feed blocks (of size at most 100) of the decoder's LSTM hidden states through a (Kim, 2014)-style convolutional model. We use kernels of width 3 and 5, 200 filters, a ReLU nonlinearity, and max-over-time pooling. To create the p_k, these now 400-dimensional features are then mapped via an MLP with a ReLU nonlinearity into 3 separate 200 dimensional vectors corresponding to the predicted relation's entity, value, and type, respectively. These 200 dimensional vectors are then fed through (separate) linear decoders and softmax layers in order to obtain distributions over entities, values, and types. We use $K = 3$ distinct p_k.

Models are trained with SGD, a learning rate of 1 (which is divided by 2 every time validation perplexity fails to decrease), and a batch size of 16. We use dropout (at a rate of 0.5) between LSTM layers and before the linear decoder.

A.3 Information Extraction Details

Data To form an information extraction dataset, we first sentence-tokenize the gold summary documents $y_{1:T}$ using NLTK (Bird, 2006). We then determine which word-spans $y_{i:j}$ could represent entities (by matching against players, teams, or cities in the database), and which word-spans $y_{k:l}$ could represent numbers (using the open source `text2num` library[1] to convert (strings of) number-words into numbers).We then consider each $y_{i:j}, y_{k:l}$ pair in the same sentence, and if there is a record r in the database such that $r.e = y_{i:j}$ and $r.m = \text{text2num}(y_{k:l})$ we annotate the $y_{i:j}, y_{k:l}$ pair with the label $r.t$; otherwise, we give it a label of ϵ.

[1] https://github.com/exogen/text2num

Model We predict relations by ensembling 3 convolutional models and 3 bidirectional LSTM (Hochreiter and Schmidhuber, 1997; Graves and Schmidhuber, 2005) models. Each model consumes the words in the sentence, which are embedded in $\mathbb{R}^{200}$, as well as the distances of each word in the sentence from both the entity-word-span and the number-word-spans (as described above), which are each embedded in $\mathbb{R}^{100}$. These vectors are concatenated (into a vector in $\mathbb{R}^{500}$) and fed into either a convolutional model or a bidirectional LSTM model.

The convolutional model uses 600 total filters, with 200 filters for kernels of width 2, 3, and 5, respectively, a ReLU nonlinearity, and max-pooling. These features are then mapped via a 1-layer (ReLU) MLP into $\mathbb{R}^{500}$, which predicts one of the 39 relation types (or ϵ) with a linear decoder layer and softmax.

The bidirectional LSTM model uses a single layer with 500 units in each direction, which are concatenated. The hidden states are max-pooled, and then mapped via a 1-layer (ReLU) MLP into $\mathbb{R}^{700}$, which predicts one of the 39 relation types (or ϵ) with a linear decoder layer and softmax.

References

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. *ACL* .

Gabor Angeli, Percy Liang, and Dan Klein. 2010. A simple domain-independent probabilistic approach to generation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 502–512.

Amit Bagga and Breck Baldwin. 1998. Algorithms for Scoring Coreference Chains. In *The first international conference on language resources and evaluation workshop on linguistics coreference*. Citeseer, volume 1, pages 563–566.

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

Regina Barzilay and Mirella Lapata. 2005. Collective content selection for concept-to-text generation. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 331–338.

Anja Belz. 2008. Automatic generation of weather forecast texts using comprehensive probabilistic generation-space models. *Natural Language Engineering* 14(04):431–455.

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

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research* 3(Feb):1137–1155.

Eric Bengtson and Dan Roth. 2008. Understanding the Value of Features for Coreference Resolution. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 294–303.

Steven Bird. 2006. Nltk: the natural language toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions*. Association for Computational Linguistics, pages 69–72.

Anders Björkelund and Richárd Farkas. 2012. Data-driven Multilingual Coreference Resolution using Resolver Stacking. In *Joint Conference on EMNLP and CoNLL-Shared Task*. Association for Computational Linguistics, pages 49–55.

Anders Björkelund and Jonas Kuhn. 2014. Learning structured perceptrons for coreference Resolution with Latent Antecedents and Non-local Features. *ACL, Baltimore, MD, USA, June* .

Léon Bottou. 1991. *UNE APPROCHE THEORIQUE DE L'APPRENTISSAGE CONNEXIONNISTE ET APPLICATIONS A LA RECONNAISSANCE DE LA PAROLE.* Ph.D. thesis.

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Józefowicz, and Samy Bengio. 2016. Generating sentences from a continuous space. In *CoNLL*. pages 10–21.

Eric Brill and Robert C Moore. 2000. An improved error model for noisy channel spelling correction. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, pages 286–293.

Jie Cai and Michael Strube. 2010. End-to-end coreference resolution via hypergraph partitioning. In *23rd International Conference on Computational Linguistics (COLING)*. pages 143–151.

Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico. 2014. Report on the 11th iwslt evaluation campaign. In *Proceedings of IWSLT, 20014*.

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daume, and John Langford. 2015. Learning to search better than your teacher. In *International Conference on Machine Learning*. pages 2058–2066.

Kai-Wei Chang, Rajhans Samdani, and Dan Roth. 2013. A Constrained Latent Variable Model for Coreference Resolution. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. pages 601–612.

Kai-Wei Chang, Rajhans Samdani, Alla Rozovskaya, Mark Sammons, and Dan Roth. 2012. Illinois-coref: The UI System in the CoNLL-2012 Shared Task. In *Joint Conference on EMNLP and CoNLL-Shared Task*. Association for Computational Linguistics, pages 113–117.

Boxing Chen and Colin Cherry. 2014. A systematic comparison of smoothing techniques for sentence-level bleu. *ACL 2014* page 362.

Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *EMNLP*. pages 740–750.

David L Chen and Raymond J Mooney. 2008. Learning to sportscast: a test of grounded language acquisition. In *Proceedings of the 25th international conference on Machine learning*. ACM, pages 128–135.

Andrew Chisholm, Will Radford, and Ben Hachey. 2017. Learning to generate one-sentence biographies from wikidata. *CoRR* abs/1702.06235.

KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation* .

Kevin Clark and Christopher D. Manning. 2015. Entity-centric coreference resolution with model stacking. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL)*. pages 1405–1415.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*. Association for Computational Linguistics, pages 1–8.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, page 111.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research* 12:2493–2537.

Aron Culotta, Michael Wick, Robert Hall, and Andrew McCallum. 2007. First-order Probabilistic Models for Coreference Resolution. In *Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics (NAACL HLT)*.

Hal Daumé III. 2017. Structured prediction is *not* rl. https://nlpers.blogspot.com/2017/04/structured-prediction-is-not-rl.html.

Hal Daumé III, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine Learning* 75(3):297–325.

Hal Daumé III and Daniel Marcu. 2005. Learning as search optimization: approximate large margin methods for structured prediction. In *Proceedings of the Twenty-Second International Conference on Machine Learning (ICML 2005)*. pages 169–176.

Harold Charles Daumé III. 2006. *Practical Structured Learning Techniques for Natural Language Processing*. Ph.D. thesis, UNIVERSITY OF SOUTHERN CALIFORNIA.

Pascal Denis and Jason Baldridge. 2008. Specialized Models and Ranking for Coreference Resolution. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 660–669.

Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. 2014. Hc-search: A learning framework for search-based structured prediction. *Journal of Artificial Intelligence Research* 50:369–407.

Cícero Nogueira dos Santos, Bing Xiang, and Bowen Zhou. 2015. Classifying relations by ranking with convolutional neural networks. In *ACL*. pages 626–634.

Pablo A Duboue and Kathleen R McKeown. 2003. Statistical acquisition of content selection rules for natural language generation. In *EMNLP*. Association for Computational Linguistics, pages 121–128.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *The Journal of Machine Learning Research* 12:2121–2159.

Greg Durrett and Dan Klein. 2013. Easy Victories and Uphill Battles in Coreference Resolution. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. pages 1971–1982.

Greg Durrett and Dan Klein. 2014. A Joint Model for Entity Analysis: Coreference, Typing, and Linking. *Transactions of the Association for Computational Linguistics* 2:477–490.

Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive Science* 14(2):179–211.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. 2010. Why does unsupervised pre-training help deep learning? *The Journal of Machine Learning Research* 11:625–660.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. Liblinear: A Library for Large Linear Classification. *The Journal of Machine Learning Research* 9:1871–1874.

Eraldo Rezende Fernandes, Cícero Nogueira Dos Santos, and Ruy Luiz Milidiú. 2012. Latent Structure Perceptron with Feature Induction for Unrestricted Coreference Resolution. In *Joint Conference on EMNLP and CoNLL-Shared Task*. Association for Computational Linguistics, pages 41–48.

Katja Filippova, Enrique Alfonseca, Carlos A Colmenares, Lukasz Kaiser, and Oriol Vinyals. 2015. Sentence compression by deletion with lstms. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. pages 360–368.

Thomas Finley and Thorsten Joachims. 2008. Training structural svms when exact inference is intractable. In *Proceedings of the 25th international conference on Machine learning*. ACM, pages 304–311.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2001. *The elements of statistical learning*, volume 1. Springer series in statistics New York.

Claudio Gentile. 2001. A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research* 2(Dec):213–242.

Eli Goldberg, Norbert Driedger, and Richard I Kittredge. 1994. Using natural-language processing to produce weather forecasts. *IEEE Expert* 9(2):45–53.

Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks* 18(5):602–610.

Barbara J Grosz. 1977. The representation and use of focus in dialogue understanding. Technical report, SRI International Menlo Park United States.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O. K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *ACL*.

Çaglar Gülçehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. 2016. Pointing the unknown words. In *ACL*.

Aria Haghighi and Dan Klein. 2007. Unsupervised coreference resolution in a nonparametric bayesian model. In *Proceedings of the 45th annual meeting of the association of computational linguistics*. pages 848–855.

Aria Haghighi and Dan Klein. 2010. Coreference Resolution in a Modular, Entity-centered Model. In *The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, pages 385–393.

Jerry R Hobbs. 1978. Coherence and coreference. Technical report, SRI INTERNATIONAL MENLO PARK CA.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.* 9:1735–1780.

Eduard Hovy, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. 2006. Ontonotes: the 90% Solution. In *Proceedings of the human language technology conference of the NAACL, Companion Volume: Short Papers*. Association for Computational Linguistics, pages 57–60.

Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, pages 142–151.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *ACL Volume 1: Long Papers*.

Dan Jurafsky and James H Martin. 2014. *Speech and language processing*, volume 3. Pearson London.

Matti Kääriäinen. 2006. Lower bounds for reductions. In *Atomic Learning Workshop*.

Anjuli Kannan and Oriol Vinyals. 2016. Adversarial evaluation of dialogue models. In *NIPS 2016 Workshop on Adversarial Training*.

Joohyun Kim and Raymond J Mooney. 2010. Generative alignment and semantic parsing for learning from ambiguous supervision. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*. Association for Computational Linguistics, pages 543–551.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *EMNLP*. pages 1746–1751.

Brian Kingsbury. 2009. Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling. In *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*. IEEE, pages 3761–3764.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. 2017. Opennmt: Open-source toolkit for neural machine translation. *CoRR* abs/1701.02810. http://arxiv.org/abs/1701.02810.

Philipp Koehn. 2004. Statistical Significance Tests for Machine Translation Evaluation. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*. Citeseer, pages 388–395.

Ioannis Konstas and Mirella Lapata. 2013. A global model for concept-to-text generation. *J. Artif. Intell. Res.(JAIR)* 48:305–346.

Karen Kukich. 1983. Design of a knowledge-based report generator. In *ACL*. pages 145–150.

Jonathan K. Kummerfeld and Dan Klein. 2013. Error-driven Analysis of Challenges in Coreference Resolution. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, WA, USA.

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 2001)*. pages 282–289.

Rémi Lebret, David Grangier, and Michael Auli. 2016. Neural text generation from structured data with application to the biography domain. In *EMNLP*. pages 1203–1213.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324.

Heeyoung Lee, Angel Chang, Yves Peirsman, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. 2013. Deterministic Coreference Resolution based on Entity-centric, Precision-ranked Rules. *Computational Linguistics* 39(4):885–916.

Heeyoung Lee, Yves Peirsman, Angel Chang, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. 2011. Stanford's Multi-pass Sieve Coreference Resolution System at the CoNLL-2011 Shared Task. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, pages 28–34.

Nicholas Léonard, Sagar Waghmare, Yang Wang, and Jin-Hwa Kim. 2015. rnn: Recurrent library for torch. *arXiv preprint arXiv:1511.07889* .

Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. 2016. Visualizing and understanding neural models in nlp. In *NAACL HLT*.

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

Percy Liang, Michael I Jordan, and Dan Klein. 2009. Learning semantic correspondences with less supervision. In *ACL*. Association for Computational Linguistics, pages 91–99.

Chia-Wei Liu, Ryan Lowe, Iulian Serban, Michael Noseworthy, Laurent Charlin, and Joelle Pineau. 2016. How NOT to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. In *EMNLP*. pages 2122–2132.

Yijia Liu, Yue Zhang, Wanxiang Che, and Bing Qin. 2015. Transition-based syntactic linearization. In *Proceedings of NAACL*.

Wei Lu and Hwee Tou Ng. 2011. A probabilistic forest-to-string model for language generation from typed lambda calculus expressions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 1611–1622.

Xiaoqiang Luo. 2005. On Coreference Resolution Performance Metrics. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 25–32.

Xiaoqiang Luo, Sameer Pradhan, Marta Recasens, and Eduard Hovy. 2014. An Extension of BLANC to System Mentions. *Proceedings of ACL, Baltimore, Maryland, June* .

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, EMNLP 2015*. pages 1412–1421.

Chao Ma, Janardhan Rao Doppa, J Walker Orr, Prashanth Mannem, Xiaoli Fern, Tom Dietterich, and Prasad Tadepalli. 2014. Prune-and-score: Learning for Greedy Coreference Resolution. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*.

Sebastian Martschat and Michael Strube. 2015. Latent structures for coreference resolution. *TACL* 3:405–418.

Andrew McCallum, Dayne Freitag, and Fernando CN Pereira. 2000. Maximum entropy markov models for information extraction and segmentation. In *Icml*. volume 17, pages 591–598.

Andrew McCallum and Ben Wellner. 2003. Toward Conditional Models of Identity Uncertainty with Application to Proper Noun Coreference. *Advances in Neural Information Processing Systems 17* .

Kathleen McKeown. 1992. *Text generation - using discourse strategies and focus constraints to generate natural language text*. Studies in natural language processing. Cambridge University Press.

Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. 2016. What to talk about and how? selective generation using lstms with coarse-to-fine alignment. In *NAACL HLT*. pages 720–730.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *CoRR* abs/1609.07843.

T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. pages 3111–3119.

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. 2012. *Foundations of machine learning*. MIT press.

Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*. pages 807–814.

Vincent Ng. 2004. Learning Noun Phrase Anaphoricity to Improve Coreference Resolution: Issues in Representation and Optimization. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, page 151.

Vincent Ng and Claire Cardie. 2002. Identifying Anaphoric and Non-anaphoric Noun Phrases to Improve Coreference Resolution. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*. Association for Computational Linguistics, pages 1–7.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics, pages 311–318.

Haoruo Peng, Kai-Wei Chang, and Dan Roth. 2015. A joint framework for coreference resolution and mention head detection. In *Proceedings of the 19th Conference on Computational Natural Language Learning (CoNLL)*. pages 12–21.

Vu Pham, Théodore Bluche, Christopher Kermorvant, and Jérôme Louradour. 2014. Dropout improves recurrent neural networks for handwriting recognition. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*. IEEE, pages 285–290.

Hoifung Poon and Pedro M. Domingos. 2008. Joint unsupervised coreference resolution with markov logic. In *2008 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. pages 650–659.

Sameer Pradhan, Xiaoqiang Luo, Marta Recasens, Eduard Hovy, Vincent Ng, and Michael Strube. 2014. Scoring Coreference Partitions of Predicted Mentions: A Reference Implementation. In *Proceedings of the Association for Computational Linguistics*.

Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Olga Uryupina, and Yuchen Zhang. 2012. Conll-2012 Shared Task: Modeling Multilingual Unrestricted Coreference in OntoNotes. In *Joint Conference on EMNLP and CoNLL-Shared Task*. Association for Computational Linguistics, pages 1–40.

Lawrence R Rabiner. 1989. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE* 77(2):257–286.

Altaf Rahman and Vincent Ng. 2009. Supervised Models for Coreference Resolution. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2*. Association for Computational Linguistics, pages 968–977.

Altaf Rahman and Vincent Ng. 2011. Narrowing the modeling gap: A cluster-ranking approach to coreference resolution. *J. Artif. Intell. Res. (JAIR)* 40:469–521.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2016. Sequence level training with recurrent neural networks. *ICLR* .

Adwait Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In *Conference on Empirical Methods in Natural Language Processing*.

Marta Recasens, Marie-Catherine de Marneffe, and Christopher Potts. 2013. The Life and Death of Discourse Entities: Identifying Singleton Mentions. In *HLT-NAACL*. pages 627–633.

Ehud Reiter. 2017. You need to understand your corpora! the weathergov example. https://ehudreiter.com/2017/05/09/weathergov/.

Ehud Reiter and Robert Dale. 1997. Building applied natural language generation systems. *Natural Language Engineering* 3(1):57–87.

Ehud Reiter, Somayajulu Sripada, Jim Hunter, Jin Yu, and Ian Davy. 2005. Choosing words in computer-generated weather forecasts. *Artificial Intelligence* 167(1-2):137–169.

Jacques Robin. 1994. *Revision-based generation of Natural Language Summaries providing historical Background*. Ph.D. thesis, Citeseer.

Stephane Ross and J Andrew Bagnell. 2014. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979* .

Stéphane Ross, Geoffrey J. 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.

Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. 2003. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Upper Saddle River.

Hasim Sak, Oriol Vinyals, Georg Heigold, Andrew W. Senior, Erik McDermott, Rajat Monga, and Mark Z. Mao. 2014. Sequence discriminative distributed training of long short-term memory recurrent neural networks. In *INTERSPEECH 2014*. pages 1209–1213.

Allen Schmaltz, Alexander M Rush, and Stuart M Shieber. 2016. Word ordering without syntax. *arXiv preprint arXiv:1604.08633* .

Iulian Vlad Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C. Courville, and Joelle Pineau. 2016. Building end-to-end dialogue systems using generative hierarchical neural network models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. pages 3776–3784.

Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. 2011. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming* 127(1):3–30.

Shiqi Shen, Yong Cheng, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. 2016. Minimum risk training for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016*.

Noah A Smith. 2011. Linguistic structure prediction. *Synthesis lectures on human language technologies* 4(2):1–274.

Noah A Smith and Mark Johnson. 2007. Weighted and probabilistic context-free grammars are equally expressive. *Computational Linguistics* 33(4):477–491.

Wee Meng Soon, Hwee Tou Ng, and Daniel Chung Yong Lim. 2001. A Machine Learning Approach to Coreference Resolution of Noun Phrases. *Computational Linguistics* 27(4):521–544.

Radu Soricut and Daniel Marcu. 2006. Stochastic language generation using widl-expressions and its application in machine translation and summarization. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, pages 1105–1112.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15(1):1929–1958.

Veselin Stoyanov and Jason Eisner. 2012. Easy-first Coreference Resolution. In *COLING*. Citeseer, pages 2519–2534.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. 2013. On the Importance of Initialization and Momentum in Deep Learning. In *Proceedings of the 30th International Conference on Machine Learning*. pages 1139–1147.

Ilya Sutskever, James Martens, and Geoffrey E Hinton. 2011. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*. pages 1017–1024.

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

Charles Sutton, Andrew McCallum, and Khashayar Rohanimanesh. 2007. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. *Journal of Machine Learning Research* 8(Mar):693–723.

Charles Sutton, Andrew McCallum, et al. 2012. An introduction to conditional random fields. *Foundations and Trends® in Machine Learning* 4(4):267–373.

Kumiko Tanaka-Ishii, Kôiti Hasida, and Itsuki Noda. 1998. Reactive content selection in the generation of real-time soccer commentary. In *COLING-ACL*. pages 1282–1288.

Ben Taskar, Carlos Guestrin, and Daphne Koller. 2004. Max-margin markov networks. In *Advances in neural information processing systems*. pages 25–32.

Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. 2005. Large margin methods for structured and interdependent output variables. *Journal of machine learning research* 6(Sep):1453–1484.

Zhaopeng Tu, Yang Liu, Lifeng Shang, Xiaohua Liu, and Hang Li. 2017. Neural machine translation with reconstruction. In *AAAI*. pages 3097–3103.

Laurens van der Maaten and Geoffrey E. Hinton. 2012. Visualizing non-metric similarities in multiple maps. *Machine Learning* 87(1):33–55.

Subhashini Venugopalan, Marcus Rohrbach, Jeffrey Donahue, Raymond J. Mooney, Trevor Darrell, and Kate Saenko. 2015. Sequence to sequence - video to text. In *ICCV*. pages 4534–4542.

Marc Vilain, John Burger, John Aberdeen, Dennis Connolly, and Lynette Hirschman. 1995. A Model-theoretic Coreference Scoring Scheme. In *Proceedings of the 6th conference on Message Understanding*. Association for Computational Linguistics, pages 45–52.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*. pages 2755–2763.

Andreas Vlachos. 2013. An investigation of imitation learning algorithms for structured prediction. In *European Workshop on Reinforcement Learning*. pages 143–154.

Paul Voigtlaender, Patrick Doetsch, Simon Wiesler, Ralf Schluter, and Hermann Ney. 2015. Sequence-discriminative training of recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, pages 2100–2104.

Taro Watanabe and Eiichiro Sumita. 2015. Transition-based neural constituent parsing. *Proceedings of ACL-IJCNLP* .

Bonnie Lynn Webber. 1978. A formal approach to discourse anaphora. Technical report, BOLT BERANEK AND NEWMAN INC CAMBRIDGE MA.

Michael White, Rajakrishnan Rajkumar, and Scott Martin. 2007. Towards broad coverage surface realization with ccg. In *Proceedings of the Workshop on Using Corpora for NLG: Language Generation and Machine Translation (UCNLG+ MT)*. pages 267–276.

Terry Winograd. 1972. Understanding natural language. *Cognitive psychology* 3(1):1–191.

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.

Sam Wiseman, Alexander M. Rush, and Stuart M. Shieber. 2016a. Antecedent prediction without a pipeline. Association for Computational Linguistics.

Sam Wiseman, Alexander M Rush, and Stuart M Shieber. 2016b. Learning global features for coreference resolution. In *Proceedings of NAACL-HLT*. pages 994–1004.

Sam Wiseman, Alexander M. Rush, Stuart M. Shieber, and Jason Weston. 2015. Learning anaphoricity and antecedent ranking features for coreference resolution. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL)*. pages 1416–1426.

Sam Wiseman, Stuart Shieber, and Alexander Rush. 2017. Challenges in data-to-document generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. pages 2253–2263.

Yuk Wah Wong and Raymond J Mooney. 2007. Generation by inverting a semantic parser that uses statistical machine translation. In *HLT-NAACL*. pages 172–179.

William A Woods, Ronald M Kaplan, and Bonnie Nash-Webber. 1972. *The lunar sciences natural language information system*. Bolt, Beranek and Newman, Incorporated.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* .

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*. pages 2048–2057.

Zichao Yang, Phil Blunsom, Chris Dyer, and Wang Ling. 2016. Reference-aware language models. *CoRR* abs/1611.01628.

Majid Yazdani and James Henderson. 2015. Incremental recurrent neural network dependency parser with search-based discriminative training. In *Proceedings of the 19th Conference on Computational Natural Language Learning, (CoNLL 2015)*. pages 142–152.

Chun-Nam John Yu and Thorsten Joachims. 2009. Learning Structural SVMs with Latent Variables. In *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, pages 1169–1176.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *CoRR* abs/1409.2329.

Daojian Zeng, Kang Liu, Siwei Lai, Guangyou Zhou, Jun Zhao, et al. 2014. Relation classification via convolutional deep neural network. In *COLING*. pages 2335–2344.

Yue Zhang and Stephen Clark. 2011. Syntax-based grammaticality improvement using ccg and guided search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 1147–1157.

Yue Zhang and Stephen Clark. 2015. Discriminative syntax-based word ordering for text generation. *Computational Linguistics* 41(3):503–538.

Zhu Zhang. 2004. Weakly-supervised relation classification for information extraction. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*. ACM, pages 581–588.

GuoDong Zhou, JunHui Li, LongHua Qian, and Qiaoming Zhu. 2008. Semi-supervised learning for relation extraction. In *Third International Joint Conference on Natural Language Processing*. page 32.

Hao Zhou, Yue Zhang, and Jiajun Chen. 2015. A neural probabilistic structured-prediction model for transition-based dependency parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*. pages 1213–1222.