ABSTRACT

Title of thesis: Discourse-Level Language Understanding with Deep Learning

Mohit Iyyer, Doctor of Philosophy, 2017

Thesis directed by: Professor Jordan Boyd-Graber
Computer Science, University of Maryland
Professor Hal Daumé III
Computer Science, University of Maryland

Designing computational models that can understand language at a human level is a foundational goal in the field of natural language processing (**NLP**). Given a sentence, machines are capable of translating it into many different languages, generating a corresponding syntactic parse tree, marking words that refer to people or places, and much more. These tasks are solved by statistical machine learning algorithms, which leverage patterns in large datasets to build predictive models. Many recent advances in **NLP** are due to deep learning models (parameterized as neural networks), which bypass user-specified features in favor of building representations of language directly from the text.

Despite many deep learning-fueled advances at the word and sentence level, however, computers still struggle to understand high-level discourse structure in language, or the way in which authors combine and order different units of text (e.g., sentences, paragraphs, chapters) to express a coherent message or narrative. Part of the reason is data-related, as there are few existing datasets for contextual language-based problems, and some tasks are too complex to be framed as supervised learning problems; for the latter type, we

must either resort to unsupervised learning or devise training objectives that simulate the supervised setting. Another reason is architectural: neural networks designed for sentence-level tasks require additional functionality, interpretability, and efficiency to operate at the discourse level. In this thesis, I design deep learning architectures for three NLP tasks that require integrating information across high-level linguistic context: question answering, fictional relationship understanding, and comic book narrative modeling. While these tasks are very different from each other on the surface, I show that similar neural network modules can be used in each case to form contextual representations. I conclude by discussing potential avenues for future research that seeks to understand increasingly large and complex context.

Discourse-Level Language Understanding with Deep Learning

by

Mohit Iyyer

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2017

Advisory Committee:
Professor Jordan Boyd-Graber (Chair, Co-Advisor)
Professor Hal Daumé III (Co-Advisor)
Professor Marine Carpuat
Professor Kyunghyun Cho
Professor Philip Resnik (Dean's Representative)

© Copyright by
Mohit Iyyer
2017

Acknowledgments

First and foremost, I would like to thank my two advisors, Jordan Boyd-Graber and Hal Daumé III, for all of their guidance throughout my time at UMD. In my first and second year when I didn't know anything, I really appreciated Jordan's patience in answering my constant stream of naïve questions. For my first conference submission, he stayed in the lab until the 3AM deadline to help me edit and polish it; the final product would have been an incoherent mess if it weren't for him. He also wrote thousands of lines of code for our initial quiz bowl project, and his dedication to constantly improving the system motivated me to work harder as well. Similarly, Hal significantly improved all of my papers and talks with his comments and edits, and I greatly enjoyed our long meetings (always scheduled for "a few minutes") that would invariably go off into tangents. Most of all, I want to thank both Jordan and Hal for the freedom they allowed me to pursue my own ideas; even when it became clear that a particular research direction was hopeless (which happened quite often!), they'd let me explore it until I was content; despite the inevitable failures, these experiences were invaluable in building both my scientific knowledge and my understanding of how to do research. I can only hope to become as great of an advisor to my own students as they've been to me.

I'd also like to thank the other members of my committee, Philip Resnik, Marine Carpuat, and Kyunghyun Cho, for their insightful comments and questions that helped me improve this document. Philip deserves special mention for his timely criticism on many of my papers and presentations, which would have otherwise been mind-numbingly dull, as does Naomi Feldman (and other members of the UMD Language Science community)

for our many educational conversations. I'd also like to thank my supervisors from two internships: Richard Socher (MetaMind), Scott Yih (Microsoft Research), and Ming-Wei Chang (Microsoft Research). In addition to being a great supervisor and collaborator, Richard's first paper on recursive neural networks inspired me to start working on deep learning, and his prompt answers to my numerous nagging questions were instrumental in helping me learn, especially since no one else at UMD was working on deep learning at that time. Similarly, Scott and Ming-Wei taught me to look more critically at research problems: behind the flashy method proposed in the latest paper there is often a simple hack that is responsible for much of the method's performance.

Of course, I couldn't have completed this journey without the other students in the CLIP lab: Jordan's senior students, Viet-An Nguyen, Yuening Hu, and Ke Zhai, who helped me tremendously when I first joined the lab (I will always remember our inadvertent hike up the Green Mountain in Boulder wearing jeans); Peter Enns and Andy Garron (our antics made coursework and day-to-day life in the lab tolerable); my collaborators, co-authors and friends Snigdha Chaturvedi, Yogarshi Vyas, He He, and Leonardo Claudino; Alvin Grissom II and our millions of lunches at Japanese restaurants; Thang Nguyen and our futile efforts to understand deep learning theory; Anupam Guha and countless rants about politics, television, and modern art; Varun Manjunatha, my main student collaborator over the years, and the countless late nights we wasted in Board & Brew working on projects going nowhere (it was worth it for the few that succeeded!); and other students in the lab and elsewhere: Sudha Rao, Bharat Singh, John Wieting, Kun Wang (and the rest of our basketball team!), Kiante Brantley, Weiwei Yang, Ahmed Elgohary, Pedro Rodriguez, Fenfei Guo, Austin Myers, Rashmi Sankepally, John Morgan, and Jyothi Krishnamurthy.

Thanks to all of my childhood and college friends, who have helped me keep my sanity during the grind of graduate school, and finally, to my family: my loving parents, who continue to encourage me to pursue my dreams, and my sister, who will always remain a beacon of reason in the sea of my ignorance (her words, not mine).

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

Designing computational models that can understand language at a human level is a foundational goal in the field of *natural language processing* (**NLP**). Given a sentence, machines are now capable of translating it into many different languages, generating a corresponding syntactic parse tree, and marking words within the sentence that refer to people or places. These tasks are solved by statistical machine learning (**ML**) algorithms, which leverage patterns in large datasets to build predictive models. Many recent advances in **NLP** are due to *deep learning* models, which distinguish themselves from other **ML** models by building representations of language directly from the text instead of relying on user-specified features. My thesis explores deep learning for language-based tasks that go *beyond* the sentence level, requiring understanding of both immediate and high-level context to solve.

Deep learning refers to a family of models called *deep neural networks* that pass their input through multiple layers, each of which performs a nonlinear transformation. In the common training paradigm of *supervised learning*, networks learn from large datasets of input/output pairs; for example, consider the task of machine translation, where neural networks reach state-of-the-art performance when trained on parallel corpora containing millions of sentence-to-sentence translations (e.g., English-to-French). The deeper and

larger the network, the more labeled examples it needs to generalize. Many low-level tasks have large annotated datasets that were created decades ago (e.g., Europarl, OntoNotes, Penn Treebank), making them popular choices for deep learning researchers.

Despite these advances at the word and sentence level, however, computers still struggle to understand high-level *discourse* structure in language, or the way in which authors combine and order different units of text (e.g., sentences, paragraphs, chapters) to express a coherent message or narrative. Part of the reason is data-related, as there are no existing datasets for many contextual language-based problems, and some tasks are too complex to be framed as supervised learning problems. For the latter type, we must either resort to unsupervised learning or devise training objectives that simulate the supervised setting.

1.2 Roadmap

In this thesis, I design deep learning architectures for three different **NLP** tasks that require integrating information across high-level linguistic contexts: question answering, fictional relationship understanding, and comic book narrative modeling. On the surface these tasks are very different from one another, and if we were to use a traditional feature-engineering based approach to solve them, each task's feature set would be very different as well. However, within the deep learning framework, the neural network architectures designed for these tasks all use the same building blocks; I will go over these basic modules in Chapter 2. The following three chapters, detailed more fully below, correspond to each of the three tasks. Finally, Chapter 7 concludes the thesis by discussing the pros and cons

of deep learning methods for large-scale language understanding and also presents future directions for this kind of research.

1.2.1 Question Answering from Paragraphs and Conversations

Computerized question answering (**QA**) is a diverse subfield of **NLP**. There are many types of questions that we may want a computer to be able to answer, of which we focus on (1) *factoid* questions, whose answers are well-known entities (e.g., trivia questions), and (2) *logical* questions, which require parsing natural language into an intermediate logical representation that is then executed on a database to retrieve the answer. While fundamentally different from one another, both of these types contain instances where discourse understanding is required to answer the question.

Chapter 3 focuses specifically on *quiz bowl*, a factoid **QA** trivia game whose questions are similar to those in the TV show "Jeopardy!". In the quiz bowl setting, questions are four to five sentences long, each of which independently identifies the answer. We formulate quiz bowl as a supervised classification problem using a large dataset of question-answer pairs; this setting is most similar to the low-level **NLP** tasks discussed earlier. Here most of our modeling is at the sentence-level, and we look at simple techniques for aggregating information across the entire question.

In Chapter 4, I look at *sequential semantic parsing*, which requires models to learn how to convert natural language questions to an SQL-like parse language. The answers to these sequential questions are located within corresponding HTML tables from Wikipedia, which act as question-specific databases. Each sequence contains multiple questions that

often require reasoning about information from previously-observed context (e.g., "What are all of the countries that participated in the 2012 Olympics? Of those, which won at least two gold medals?"). Unlike the quiz bowl case, we do not have full supervision for this task; thus, we treat it as a structured prediction problem solved using a reward-guided search process reminiscent of reinforcement learning methods.

1.2.2 Understanding Dynamic Fictional Relationships in Novels

Both of the QA tasks involve relatively short contexts, but in many language domains human readers must make sense of large, complex text. Consider novels, which contain character-centric narratives; relationships between characters develop chapter by chapter, and events in the story often have huge impact on these relationships. Developing computer models for understanding stories is a far cry from sentential NLP tasks as it requires integrating information across huge contexts. Furthermore, the task requires processing dependencies that span thousands of sentences or more, not just a paragraph as in the QA tasks.

In Chapter 5, I design a new deep neural network architecture for temporally modeling fictional relationships across entire books. The network is trained in an unsupervised fashion, as there is no existing annotated dataset for this task and creating one is not possible given the subjective nature of the task. Since evaluation is a challenge without ground-truth annotations, I propose a novel interpretable dictionary layer whose output can be directly analyzed by human evaluators.

1.2.3 Making Connective Inferences in Comic Books

In the real world, humans process many different sources of information, not just text. For example, comic books have panels that contain both visual information (in the form of artwork) and language by way of dialogue. Stories are told in sequences of panels; to understand the action and dialogue in a particular panel, readers must have understood everything that happened in the preceding panels. One interesting aspect of comics is that their artists cannot possibly draw every action of every scene due to the physical constraints of the page. In practice, this limitation manifests itself in adjacent panels that often differ wildly in terms of space and time. Thus, integrating information across comic book panels is more difficult than in video frames, as readers must make many connective inferences to form a coherent story. In Chapter 6, I focus on computationally modeling these inferences by proposing several tasks that test a neural network's ability to predict aspects of future panels given some context.

Chapter 2

Background: Deep Learning for Discourse-Level NLP

In traditional machine learning models for NLP, humans inject substantial prior knowledge of the task into the algorithmic design in the form of *features*. In text classification problems, a classifier is then trained over these features to predict a given label. As a running example for this chapter, I consider the task of detecting the political ideology of an author (e.g., liberal or conservative, in United States politics) based on a sentence they wrote.[1] For this task, our features could be counts of words or phrases that we deem indicative of political ideology; for example, the phrase *small business* leans conservative, while *climate change* leans liberal.

Manually designing features is a fine strategy if we assume that the input sentences are simple. However, this assumption is often not valid; consider the phrase *so-called "climate change"*. It becomes difficult to manually design features that accurately represent the content of more complex constructions such as the scare quotes in this one, which flip the polarity of the phrase from liberal to conservative. Do we want to have an individual feature for *so-called "climate change"*, or perhaps make a separate feature for words in scare quotes? If the latter, how do we detect which quotation marks are used as scare quotes and which are not?

Deep learning removes the human burden of feature engineering by bypassing predefined features (or feature templates) in favor of *learned* representations from the

[1] I built a recursive neural network for this task in Iyyer et al. (2014b).

raw text input. In **NLP**, this process involves learning a *composition function*, which tells the model how to combine individual words to obtain useful representations of sentences or larger units of text. In this chapter, I introduce different neural network architectures for performing composition; the models discussed here will be expanded upon in future chapters. Then, I describe how neural models (as well as traditional **NLP** methods) have previously been used for discourse-level language comprehension, which is the topic of this thesis.

2.1 Representing Words with Vectors

Setting aside deep learning for the moment, let us consider a standard bag-of-unigrams classifier for our ideology problem. Here, we have a separate feature for each word in our vocabulary, which means that each word has a "one-hot" vector representation. The corresponding vector for a given word is of the size of the vocabulary (usually thousands of words) where each dimension k represents the identity of a distinct word w_k. For any w_k, the value at dimension k is 1, while all other dimensions are zeros.

A major problem with one-hot vectors is that they cannot by themselves give an idea of the similarity between two words, as all words are equally different from each other since the inner product between the vectors for any two words is zero. Low-dimensional *embeddings* address this issue by restricting the number of dimensions to a hyperparameter d (usually between 50 and 500) and allowing each dimension to take on a real value, as opposed to a binary zero or one. As shown in Figure 2.1, synonymous and similar words form clusters in the learned vector space when trained to capture word co-occurrence

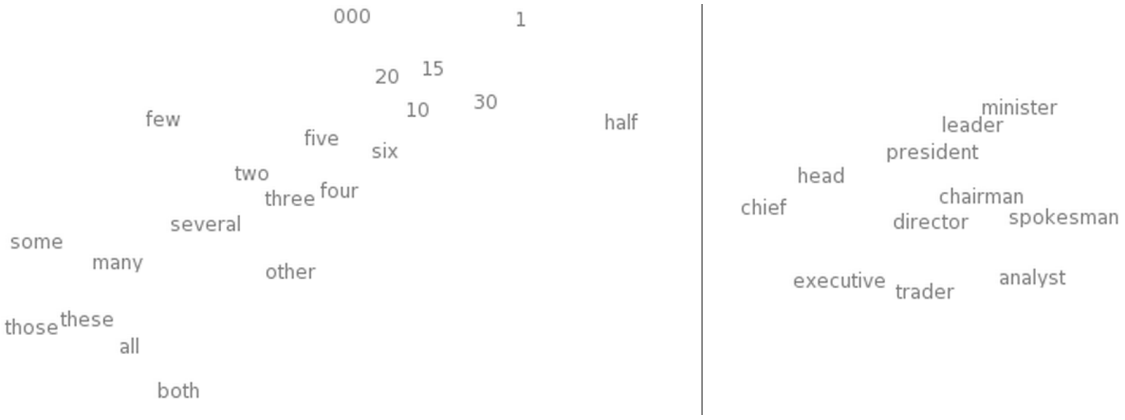


Figure 2.1: Two dimensional visualization of word embeddings. On the left side, words associated with numbers or counting are clustered together, while on the right side we see an occupation cluster. (Credit: Christopher Olah)

statistics of large datasets. These representations form the basis for the neural network architectures that I discuss in the rest of this chapter.

2.2 Simple Composition Functions: Neural Bag-of-Words

A composition function takes as input a sequence of word embeddings X and outputs a single vector $\boldsymbol{z}$. To be more concrete, we define the word embedding matrix for a given vocabulary V as $\mathbf{L}$, which is of size $d \times |V|$. Each column $\boldsymbol{v}_w$ of $\mathbf{L}$ is the embedding for the word w corresponding to that column. The input to a composition function g is then the sequence of word embeddings $\boldsymbol{v}_w$ for $w \in X$, and its output $\boldsymbol{z}$ is not necessarily of dimensionality d.

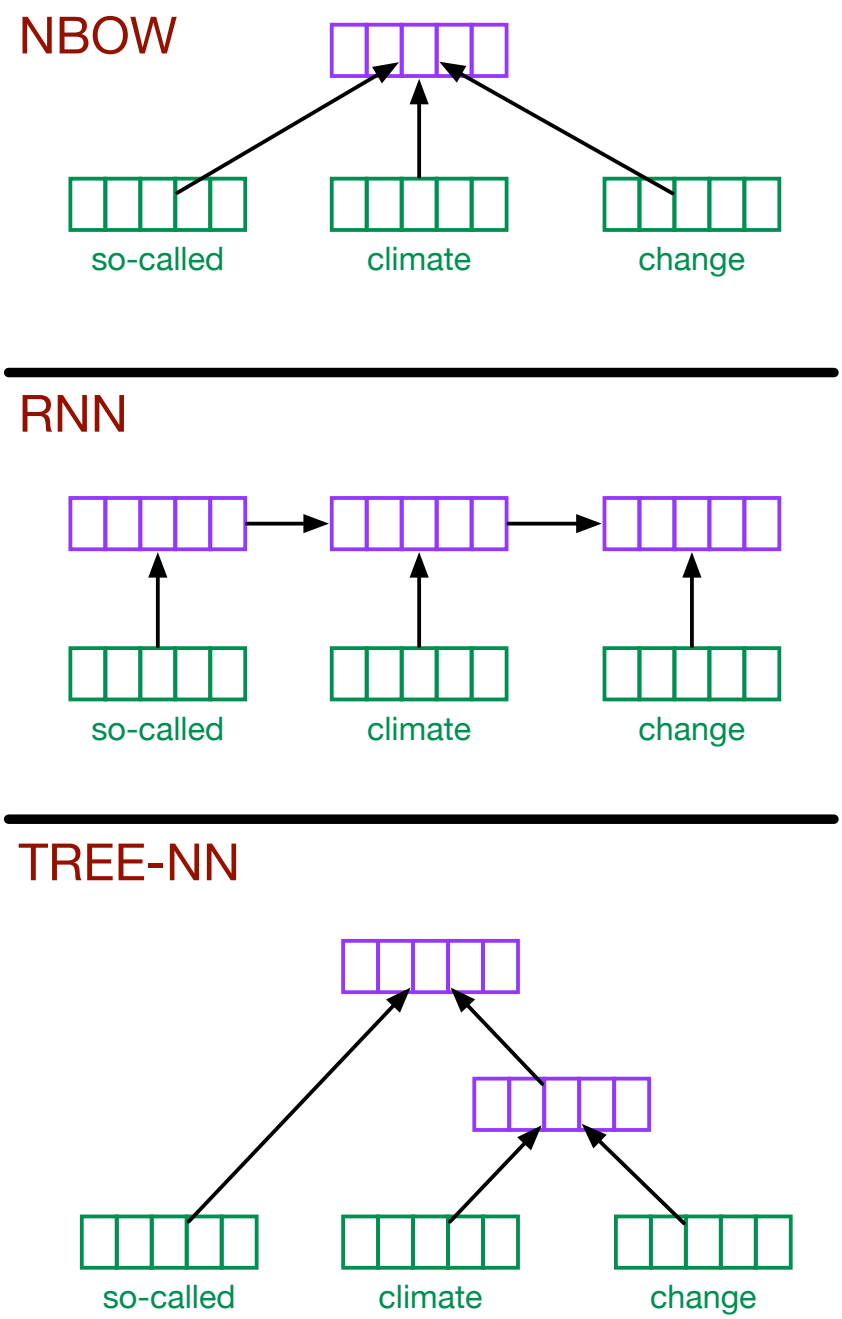


Figure 2.2: Three different neural network architectures used as composition functions for the phrase "so-called climate change". Top: neural bag-of-words; Middle: recurrent neural network; Bottom: tree-structured neural network.

The simplest composition function is the neural bag-of-words model (**NBOW**, top of Figure 2.2), which is an element-wise vector arithmetic operation that has no additional

parameters beyond L. "Bag-of-words" refers to the fact that the function does not consider the order of the words in X when computing $\boldsymbol{z}$; this is not the case with the more complex models discussed later in this chapter. Popular choices for **NBOW** functions include the *vector sum* or *vector average*; the latter can be written as

$$\boldsymbol{z} = g(w \in X) = \frac{1}{|X|} \sum_{w \in X} \boldsymbol{v}_w. \tag{2.1}$$

2.2.1 Training Models for Text Classification

How do we train our composition function to learn representations useful for, say, our ideology classification task? We can formalize the task as a binary classification problem, where we map an input sequence of tokens X to one of two labels. Since this formalism implies a supervised learning objective, we must have a dataset of sentences that have been annotated with their authors' ideologies.[2] Our goal is then to learn a composition function that does a good job of predicting these labels. Using the vector average as our composition function, we first apply Equation 2.1 to X to get a vector $\boldsymbol{z}$, which will serve as input to a classification layer.

Specifically, we apply a logistic regression that takes in $\boldsymbol{z}$ and produces a prediction $\hat{y}$. This is a softmax layer, which induces estimated probabilities for each output label:

$$\hat{y}_p = \text{softmax}(\mathbf{W}\boldsymbol{z}), \tag{2.2}$$

[2] In Iyyer et al. (2014b) I introduce the Ideological Books Corpus, a dataset for political ideology classification.

where the softmax function is

$$\text{softmax}(\boldsymbol{q}) = \frac{\exp \boldsymbol{q}}{\sum_{j=1}^{k} \exp \boldsymbol{q}_j} \tag{2.3}$$

and $\mathbf{W}$ is a $2 \times d$ matrix for our two-label problem.[3]

We want the predictions of the softmax layer to match our annotated data; the discrepancy between categorical predictions and annotations is measured through the cross-entropy loss. We optimize the model parameters to minimize the cross-entropy loss over all sentences in the corpus. The cross-entropy loss of a single example X with ground-truth label y is

$$\ell(\hat{y}) = \sum_{p=1}^{2} y_p * log(\hat{y}_p). \tag{2.4}$$

This induces a supervised objective function over all sentences in the dataset: a regularized sum over all losses normalized by the number of training examples N,

$$C = \frac{1}{N} \sum_{i}^{N} \ell(\hat{y}_i) + \frac{\lambda}{2} |\theta|^2. \tag{2.5}$$

The model parameters θ (here the word embedding matrix $\mathbf{L}$ and the softmax matrix W) can be optimized using a variety of online or batch methods. Commonly-used algorithms include Adam (Kingma and Ba, 2014) and the diagonal variant of AdaGrad (Duchi et al., 2011). The gradient of the objective, shown in Eq. (2.6), is computed using back-propagation (Rumelhart et al., 1986), which essentially involves repeated application of

[3] Bias terms should be included, but I omit them here for simplicity.

the chain rule.

$$\frac{\partial C}{\partial \theta} = \frac{1}{N} \sum_{i}^{N} \frac{\partial \ell(\hat{y}_i)}{\partial \theta} + \lambda\theta. \tag{2.6}$$

2.3 Deep NBOW Models

The NBOW is a linear model; it does not have any nonlinear transformations between its input and the classification layer, which limits its expressivity. The intuition behind deep feed-forward neural networks is that each layer learns a more abstract representation of the input than the previous one (Bengio et al., 2013). We can apply this concept to the NBOW model discussed above with the expectation that each layer will increasingly magnify small but meaningful differences in the word embedding average. To be more concrete, take s_1 as the sentence "I am supportive of legislation that aims to reduce global warming" and generate s_2 and s_3 by replacing "supportive" with "indifferent" and then again by "opposed". The vector averages of these three sentences are almost identical, but the averages associated with the synonymous sentences s_1 and s_2 are slightly more similar to each other than they are to s_3's average.

Could adding depth to NBOW make small such distinctions as this one more apparent? In Equation 2.1, we compute $\boldsymbol{z}$, the vector representation for input text X, by averaging the word vectors $\boldsymbol{v}_{w \in X}$. Instead of directly passing this representation to an output layer, we can further transform $\boldsymbol{z}$ by adding more layers before applying the softmax. Suppose we have n layers, $\boldsymbol{z}_{1...n}$. We compute each layer

$$\boldsymbol{z}_i = f(\mathbf{W}_i \cdot \boldsymbol{z}_{i-1}) \tag{2.7}$$

and feed the final layer's representation, $\boldsymbol{z}_n$, to a softmax layer for prediction. f here is an element-wise nonlinearity such as $\tanh$.

This model, which I call the deep averaging network (**DAN**), is still a bag-of-words model, but its depth allows it to capture subtle variations in the input better than the standard **NBOW** model.[4] Furthermore, computing each layer requires just a single matrix multiplication, so the complexity scales with the number of layers instead of the number of words in the sentence as in more complex models. In practice, we find no significant difference between the training time of a **DAN** and that of the shallow **NBOW** model.

2.4 Recurrent Neural Networks

Now that we know how to train simple composition functions, we move to models that consider the word order and syntactic structure of their input. We will start with the recurrent neural network, or **RNN**. Just like before, feeding a sequence of text X into an **RNN** yields a vector that represents the sequence; we will call this vector a hidden state, or $\boldsymbol{h}_n$. A softmax layer over $\boldsymbol{h}_n$ predicts the output label.

RNNs read input sentences from left to right, which means that they compute a hidden state at every word in the input. The computation of a hidden state $\boldsymbol{h}_t$ depends on both the current input embedding $\boldsymbol{v}_{w_t}$ and the previous hidden state $\boldsymbol{h}_{t-1}$. Formally, **RNN**s are defined as

$$\boldsymbol{h}_t = f(\mathbf{W} \begin{bmatrix} \boldsymbol{v}_{w_t} \\ \boldsymbol{h}_{t-1} \end{bmatrix}), \tag{2.8}$$

[4] This statement is corroborated by experimental findings in (Iyyer et al., 2015).

where, like before, f is an element-wise nonlinearity such as $\tanh$. The model is then trained just like before, except here we have an additional model parameter (the transition matrix $\mathbf{W}$), and the gradients are computed through a variant of backpropagation called *backpropagation through time* (Werbos, 1990).

RNNs can suffer from the *vanishing gradient* problem, which refers to the phenomenon of gradients from earlier time steps having much lower magnitudes than those at more recent ones. This issue arises from repeated applications of the weight matrix $\mathbf{W}$ in Equation 2.8, which is proven to result in vanishing gradients if the largest eigenvalue of $\mathbf{W}$ is less than 1 (Pascanu et al., 2013). Multiple variants have been proposed to combat this problem, the most popular of which are the long short-term memory (Hochreiter and Schmidhuber, 1997, **LSTM**) and the gated recurrent unit (Cho et al., 2014, **GRU**). I will return to **LSTM** units in Chapter 6.

2.4.1 Tree-Structured Variants

Equation 2.8 implies a fixed left-branching tree structure. The equation can, however, be extended to operate on predefined tree structures, as in the case of tree neural networks (**TreeNN**). These models were first introduced by Pollack (1990) and recently repopularized by Richard Socher and colleagues (Socher et al., 2011b,a). Here, the composition function g depends on a *syntactic parse tree* of the input sequence. The representation for any internal node in a binary parse tree is computed as a nonlinear function of the representations of its children. A more powerful **TreeNN** variant is the recursive neural tensor network (**RecNTN**), which modifies g to include a costly tensor product (Socher

et al., 2013).

Given a binary constituency parse tree of a sentence, the leaves of the tree will be the words of the sentence $w_{1...n}$ where each word is associated with a vector $\boldsymbol{v}_w \in \mathbf{L}$. The parse tree defines how these words form phrases (bottom of Figure 2.2). Each of these phrases p also has an associated vector $\boldsymbol{x}_p \in \mathbb{R}^d$ of a user-specified dimensionality. These phrase vectors should represent the meaning of the phrases composed of individual words. As phrases themselves merge into complete sentences, the underlying vector representation is trained to retain the sentence's whole meaning.

The challenge is to describe how vectors combine to form complete representations. If two words a and b merge to form phrase p, we posit that the phrase-level vector is

$$\boldsymbol{x}_p = f(\mathbf{W} \begin{bmatrix} \boldsymbol{v}_a \\ \boldsymbol{v}_b \end{bmatrix}), \tag{2.9}$$

where $\mathbf{W}$ is a $2d \times d$ composition matrix shared across all nodes in the tree.

The "recursive" aspect of the **TreeNN** lies in its use of *weight sharing*. The composition matrix is repeatedly applied to nodes in the parse tree in a bottom-up fashion. Thus, once x_p is computed for a non-root node p, it will be fed into Eq. (2.9) again as part of the computation for its parent's phrase vector $\boldsymbol{x}_{par}$:

$$\boldsymbol{x}_{par} = f(\mathbf{W} \begin{bmatrix} \boldsymbol{v}_c \\ \boldsymbol{x}_p \end{bmatrix}). \tag{2.10}$$

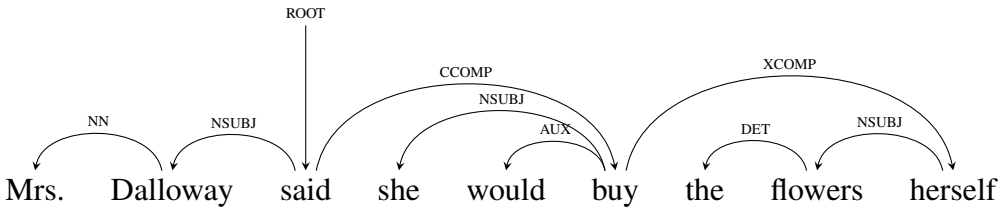


Figure 2.3: Dependency parse tree of the opening sentence of Virginia Woolf's *Mrs. Dalloway*.

2.4.1.1 Dependency Tree RecNNs

TreeNNs are not limited to only binary tree structures; in fact, dependency tree neural networks (**DTreeNN**) have shown improved performance on image-to-text mapping and question answering tasks (Socher et al., 2014; Iyyer et al., 2014a) compared to constituency tree **TreeNN**s. Because every node of a dependency parse tree is associated with a word, it has less nodes in total than a corresponding constituency tree, which reduces vanishing gradient issues. However, internal nodes of dependency parse trees are unbounded in the number of children they can have, which makes weight sharing more complicated than in the constituency case. The solution is to "untie" the composition matrices by dependency relations, which allows us to inject more syntactic information into the model.

Untying the model in this way requires more than one composition matrix, unlike the binary tree case. In particular, we associate a separate $d \times d$ matrix $\mathbf{W}_r$ with each dependency relation r in our dataset and learn these matrices during training. Syntactically untying these matrices allows the model to take advantage of relation identity as well as tree structure. We include an additional $d \times d$ matrix, $\mathbf{W}_v$, to incorporate the word vector $\boldsymbol{v}_w$ at a node n into the hidden vector $\boldsymbol{h}_n$.

Given a dependency parse tree, we first compute hidden representations for the leaf

nodes. For example, the hidden representation $\boldsymbol{h}_{\text{mrs.}}$ for the parse tree given in Figure 2.4.1.1 is

$$\boldsymbol{h}_{\text{mrs.}} = f(\mathbf{W}_v \cdot \boldsymbol{v}_{\text{mrs.}}). \tag{2.11}$$

After finishing with the leaves, we move to interior nodes whose children have already been processed. Continuing from *mrs.* to its parent, *dalloway*, we compute

$$\boldsymbol{h}_{\text{dalloway}} = f(\mathbf{W}_{\text{NN}} \cdot \boldsymbol{h}_{\text{mrs.}} + \mathbf{W}_v \cdot \boldsymbol{v}_{\text{dalloway}}). \tag{2.12}$$

We repeat this process up to the root, which is

$$\boldsymbol{h}_{\text{said}} = f(\mathbf{W}_{\text{NSUBJ}} \cdot \boldsymbol{h}_{\text{dalloway}} + \mathbf{W}_{\text{CCOMP}} \cdot \boldsymbol{h}_{\text{buy}} + \mathbf{W}_v \cdot \boldsymbol{v}_{\text{said}}). \tag{2.13}$$

The composition equation for any node n with children $K(n)$ and word vector $\boldsymbol{v}_w$ is $\boldsymbol{h}_n =$

$$f(\mathbf{W}_v \cdot \boldsymbol{v}_w + b_1 + \sum_{k \in K(n)} \mathbf{W}_{R(n,k)} \cdot \boldsymbol{h}_k), \tag{2.14}$$

where $R(n, k)$ is the dependency relation between node n and child node k. The model is trained using the same procedure as that for constituency-tree **TreeNN**s.

2.5 Discourse-Level Representation Learning

The neural network architectures described above have mainly been applied to sentence-level **NLP** tasks. The objective of this thesis is to go beyond the sentence to discourse-level language comprehension. In this section, I provide an overview of existing discourse-level

language understanding methods, and I also go over more recent deep learning-based developments in this area. The latter discussion will be at a relatively high level, as specific instances of these models will be described more fully in later chapters.

2.5.1 Discourse-Level Understanding

The term *discourse* refers to any unit of text that consists of more than one sentence. Discourse *coherence* refers to the logical order of sentences and how each fits with the others to produce a larger understandable meaning. As an example, a paragraph constructed by randomly selecting four sentences from this chapter would almost certainly have low coherence. There are many ways to formalize coherence, such as Hobbs' set of coherence relations (Hobbs, 1979) that categorizes inter-sentence relationships as "explanations", "elaborations", or "results", among others. A popular formalism in the **NLP** community is rhetorical structure theory (RST), which categorizes units of text (not just sentences) into different roles (e.g., "background", "evidence") that define how they can be combined together in a hierarchical fashion (Mann and Thompson, 1988). The task of *discourse parsing* (Soricut and Marcu, 2003; Feng and Hirst, 2012; Ji and Eisenstein, 2014) automatically discovers these sorts of relations in text, trained using annotated datasets such as the RST Discourse Treebank (Carlson et al., 2003) or the Penn Discourse Treebank (Miltsakaki et al., 2004). Other research in this vein looks at cross-sentential coreference resolution (van Hoek, 1997).

Aside from coherence, sentence *cohesion* is also an important property of discourse. Take the task of *discourse segmentation* (Passonneau and Litman, 1997), which involves

detecting boundaries in a given document where the topic of discussion shifts. Instances of *lexical cohesion* are useful for this task; for example, a repetition of certain words or their synonyms likely indicates that the topic has not changed (Halliday and Hasan, 1976). These types of features are taken into account by methods such as TextTiling (Hearst, 1997), which automatically segment documents into their subtopics.

In my thesis, I focus on coherence relations between sentences and larger units of text rather than cohesive ones. In contrast to traditional discourse research on coherence, the work in my thesis does not focus on discovering specific connections between nearby sentences (e.g., discourse relations). Instead, I focus on learning *representations* from discourse-level language that are useful for downstream tasks; this is an atheoretical approach that does not depend on a specified discourse formalism. Examples of simple and general discourse-level representations are bag-of-words (or bag-of-ngrams) vectors, or the slightly more involved TF-IDF formulation (Salton and McGill, 1986) popular in information retrieval. However, these representations are constant across all tasks, unlike machine learning models whose parameters are updating using a task-specific error signal. More complex models that perform some sort of matrix factorization, such as supervised latent Dirichlet allocation (SLDA) (Mcauliffe and Blei, 2008), are able to modify their representations from downstream supervision. Finally, research on scripts, or sequenced actions for common situations, is also relevant to the portions of my dissertation research that focus on creative language; Section 5.6 discusses this line of work in more detail.

2.5.2 Deep Discourse-Level Representation Learning

Unsupervised deep learning models have shown increased effectiveness over the above non-neural methods in terms of learning general purpose discourse-level representations. Two popular examples are Paragraph Vector (Le and Mikolov, 2014), which applies algorithms from word embedding learning to paragraphs, and recurrent autoencoder-based methods that reconstruct entire documents through bottleneck representations learned by neural networks (Li et al., 2015).

Task-specific discourse-level representations have also seen increased research interest. For example, recurrent / convolutional network hybrids have been used for document-level sentiment analysis (Tang et al., 2015), and some reading comprehension-style question-answering tasks also rely on similar methods for representing passages (Dhingra et al., 2017); these tasks and models are most related to the **QA** tasks I discuss in Chapters 3 and 4. However, building deep discourse-level models for underspecified tasks, such as the fictional relationship analysis of Chapter 5, steps farther from the current trends in this research area, and interpretability becomes a critical factor for these tasks.

Chapter 3

Deep Learning for Multi-Sentence Factoid Question Answering

Armed with both deep learning fundamentals and descriptions of both existing (**RNN**) and newly-proposed (**DAN**) models from the last chapter, we are now prepared to jump into applications of these models.[1] Here we focus on the task of *question answering* (**QA**) where input questions are multiple sentences long. How do we apply models like **TreeNN**s across multiple sentences? Are such complex models even necessary? In this chapter, I demonstrate that a very simple element-wise vector average is effective for a factoid **QA** task called *quiz bowl*, which is a trivia game whose questions describe famous entities (e.g., authors, battles, novels). The following chapter describes a more challenging **QA** task for which averaging is insufficient to capture all of the necessary contextual information, motivating the use of structured prediction methods.

3.1 Quiz Bowl: Factoid **QA**

Consider factoid question answering: given a description of an entity, identify the person, place, or thing discussed. We describe a task with high-quality mappings from natural language text to entities in Section 3.1.1. This task—quiz bowl—is a challenging natural language problem with large amounts of diverse and compositional data.

To answer quiz bowl questions, we develop a dependency tree neural network in

[1] This chapter synthesizes work previously published in Iyyer et al. (2014a, EMNLP) and Iyyer et al. (2015, ACL).

Later in its existence, this polity's leader was chosen by a group that included three bishops and six laymen, up from the seven who traditionally made the decision. Free imperial cities in this polity included Basel and Speyer. Dissolved in 1806, its key events included the Investiture Controversy and the Golden Bull of 1356. Led by Charles V, Frederick Barbarossa, and Otto I, for 10 points, name this polity, which ruled most of what is now Germany through the Middle Ages and rarely ruled its titular city.

Figure 3.1: An example quiz bowl question about the Holy Roman Empire. The first sentence contains no words or named entities that by themselves are indicative of the answer, while subsequent sentences contain more and more obvious clues.

Section 3.1.2 and extend it to combine predictions across sentences to produce a question answering neural network with trans-sentential averaging (QANTA). We evaluate our model against strong computer and human baselines in Section 3.1.3 and conclude by examining the latent space and model mistakes.

3.1.1 Matching Text to Entities: Quiz Bowl

Every weekend, hundreds of high school and college students play a game where they map raw text to well-known entities. This is a trivia competition called *quiz bowl*. Quiz bowl questions consist of four to six sentences and are associated with factoid answers (e.g., history questions ask players to identify specific battles, presidents, or events). Every sentence in a quiz bowl question is guaranteed to contain clues that uniquely identify its answer, even without the context of previous sentences. Players answer at any time—ideally more quickly than the opponent—and are rewarded for correct answers.

Automatic approaches to quiz bowl based on existing NLP techniques are doomed to failure. Quiz bowl questions have a property called *pyramidality*, which means that sentences early in a question contain harder, more obscure clues, while later sentences are "giveaways". This design rewards players with deep knowledge of a particular subject

and thwarts bag of words methods. Sometimes the first sentence contains no named entities—answering the question correctly requires an actual understanding of the sentence (Figure 3.1). Later sentences, however, progressively reveal more well-known and uniquely identifying terms.

Previous work answers quiz bowl questions using a bag of words (naïve Bayes) approach (Boyd-Graber et al., 2012). These models fail on sentences like the first one in Figure 3.1, a typical hard, initial clue. Tree-structured neural networks (**TreeNN**s), in contrast to simpler models, can capture the compositional aspect of such sentences (Hermann et al., 2013).

TreeNNs require many redundant training examples to learn meaningful representations, which in the quiz bowl setting means we need multiple questions about the same answer. Fortunately, hundreds of questions are produced during the school year for quiz bowl competitions, yielding many different examples of questions asking about any entity of note (see Section 3.1.3.1 for more details). Thus, we have built-in redundancy (the number of "askable" entities is limited), but also built-in diversity, as difficult clues cannot appear in every question without becoming well-known.

3.1.2 Negative Sampling Instead of Softmax

In this chapter we use the dependency-tree **TreeNN** (**DTreeNN**) that we defined formally in Chapter 2, with one important difference: we no longer use a softmax output layer. Our goal is to map questions to their corresponding answer entities. Because there are a limited number of possible answers, we can view this as a multi-class classification task. While a

softmax layer over every node in the tree could predict answers (Socher et al., 2011b; Iyyer et al., 2014b), this method overlooks that most answers are themselves words (features) in other questions (e.g., a question on *World War II* might mention the *Battle of the Bulge* and vice versa). Thus, word vectors associated with such answers can be trained in the same vector space as question text,[2] enabling us to model relationships between answers instead of assuming incorrectly that all answers are independent. To take advantage of this observation, we train both the answers and questions jointly in a single model, rather than training each separately and holding embeddings fixed during **DTreeNN** training.

Intuitively, we want to encourage the vectors of question sentences to be near their correct answers and far away from incorrect answers. We accomplish this goal by using a contrastive max-margin objective function described below. While we are not interested in obtaining a ranked list of answers,[3] we observe better performance by adding the weighted approximate-rank pairwise (WARP) loss proposed in Weston et al. (2011) to our objective function.

Given a sentence paired with its correct answer c, we randomly select j incorrect answers from the set of all incorrect answers and denote this subset as Z. Since c is part of the vocabulary, it has a vector $x_c \in L$. An incorrect answer $z \in Z$ is also associated with a vector $\boldsymbol{x}_z \in L$. We define S to be the set of all nodes in the sentence's dependency tree, where an individual node $s \in S$ is associated with the hidden vector $\boldsymbol{h}_s$. The error for the

[2] Of course, questions never contain their own answer as part of the text.

[3] In quiz bowl, all wrong guesses are equally detrimental to a team's score, no matter how "close" a guess is to the correct answer.

sentence is

$$C(S,\theta) = \sum_{s \in S} \sum_{z \in Z} R(rank(c,s,Z)) \max(0, 1 - \boldsymbol{x}_c \cdot h_s + x_z \cdot \boldsymbol{h}_s), \tag{3.1}$$

where the function $rank(c, s, Z)$ provides the rank of correct answer c with respect to the incorrect answers Z. We transform this rank into a loss function[4] shown by Usunier et al. (2009) to optimize the top of the ranked list, $R(r) = \sum_{i=1}^{r} 1/i$.

Since $rank(c, s, Z)$ is expensive to compute, we approximate it by randomly sampling K incorrect answers until a violation is observed ($\boldsymbol{x}_c \cdot \boldsymbol{h}_s < 1 + \boldsymbol{x}_z \cdot \boldsymbol{h}_s$) and set $rank(c, s, Z) = (|Z| - 1)/K$, as in previous work (Weston et al., 2011; Hermann et al., 2014). The model minimizes the sum of the error over all sentences T normalized by the number of nodes N in the training set,

$$J(\theta) = \frac{1}{N} \sum_{t \in T} C(t, \theta). \tag{3.2}$$

The parameters $\theta = (W_{r \in R}, W_v, W_e, b)$, where R represents all dependency relations in the data, are optimized using AdaGrad as before (Duchi et al., 2011).[5] In Section 3.1.3 we compare performance to an identical model (FIXED-QANTA) that excludes answer vectors from $\mathbf{L}$ and show that training them as part of θ produces significantly better results.

[4] Adding this loss term to the objective function not only increases performance but also speeds up convergence

[5] We set the initial learning rate $\eta = 0.05$ and reset the squared gradient sum to zero every five epochs.

The gradient of the objective function,

$$\frac{\partial C}{\partial \theta} = \frac{1}{N} \sum_{t \in T} \frac{\partial J(t)}{\partial \theta}, \tag{3.3}$$

is again computed using backpropagation through structure (Goller and Kuchler, 1996).

3.1.3 Experiments

We compare the performance of QANTA against multiple strong baselines on two datasets. QANTA outperforms all baselines trained only on question text and improves an information retrieval model trained on all of Wikipedia. QANTA requires that an input sentence describes an entity without mentioning that entity, a constraint that is not followed by Wikipedia sentences.[6] While IR methods can operate over Wikipedia text with no issues, we show that the representations learned by QANTA over just a dataset of question-answer pairs can significantly improve the performance of IR systems.

3.1.3.1 Datasets

We evaluate our algorithms on a corpus of over 100,000 question/answer pairs from two different sources. First, we expand the dataset used in Boyd-Graber et al. (2012) with publically-available questions from quiz bowl tournaments held after that work was published. This gives us 46,842 questions in fourteen different categories. To this dataset we add 65,212 questions from NAQT, an organization that runs quiz bowl tournaments and

[6] We tried transforming Wikipedia sentences into quiz bowl sentences by replacing answer mentions with appropriate descriptors (e.g., "Joseph Heller" with "this author"), but the resulting sentences suffered from a variety of grammatical issues and did not help the final result.

generously shared with us all of their questions from 1998–2013.

Because some categories contain substantially fewer questions than others (e.g., astronomy has only 331 questions), we consider only literature and history questions, as these two categories account for more than 40% of the corpus. This leaves us with 21,041 history questions and 22,956 literature questions.

Data Preparation To make this problem feasible, we only consider a limited set of the most popular quiz bowl answers. Before we filter out uncommon answers, we first need to map all raw answer strings to a canonical set to get around formatting and redundancy issues. Most quiz bowl answers are written to provide as much information about the entity as possible. For example, the following is the raw answer text of a question on the Chinese leader Sun Yat-sen: *Sun Yat-sen; or Sun Yixian; or Sun Wen; or Sun Deming; or Nakayama Sho; or Nagao Takano*. Quiz bowl writers vary in how many alternate acceptable answers they provide, which makes it tricky to strip superfluous information from the answers using rule-based approaches.

Instead, we use Whoosh,[7] an information retrieval library, to generate features in an active learning classifier that matches existing answer strings to Wikipedia titles. If we are unable to find a match with a high enough confidence score, we throw the question out of our dataset. After this standardization process and manual vetting of the resulting output, we can use the Wikipedia page titles as training labels for the **DTreeNN** and baseline models.[8]

Quiz bowl answer distribution has a long-tail: 65.6% of answers only occur once or

[7] https://pypi.python.org/pypi/Whoosh/

[8] Code and non-NAQT data available at http://cs.umd.edu/~miyyer/qblearn.

twice in the corpus. We filter out all answers that do not occur at least six times, which leaves us with 451 history answers and 595 literature answers that occur on average twelve times in the corpus. These pruning steps result in 4,460 usable history questions and 5,685 literature questions. While ideally we would have used all answers, our model benefits from many training examples per answer to learn meaningful representations; this issue can possibly be addressed with techniques from zero shot learning (Palatucci et al., 2009; Pasupat and Liang, 2014), which we leave to future work.

We apply basic named entity recognition (NER) by replacing all occurrences of answers in the question text with single entities (e.g., *Ernest Hemingway* becomes *Ernest_Hemingway*). While we experimented with more advanced NER systems to detect non-answer entities, they could not handle multi-word named entities like the book *Love in the Time of Cholera* (title case) or battle names (e.g., *Battle of Midway*). A simple search/replace on all answers in our corpus works better for multi-word entities.

The preprocessed data are split into folds by tournament. We choose the past two national tournaments[9] as our test set as well as questions previously answered by players in Boyd-Graber et al. (2012) and assign all other questions to train and dev sets. History results are reported on a training set of 3,761 questions with 14,217 sentences and a test set of 699 questions with 2,768 sentences. Literature results are reported on a training set of 4,777 questions with 17,972 sentences and a test set of 908 questions with 3,577 sentences.

Finally, we initialize the word embedding matrix W_e with word2vec (Mikolov et al., 2013) trained on the preprocessed question text in our training set.[10] We use the hierarchical

[9]The tournaments were selected because NAQT does not reuse any questions or clues within these tournaments.

[10]Out-of-vocabulary words from the test set are initialized randomly.

skip-gram model setting with a window size of five words.

3.1.4 Baselines

We pit QANTA against two types of baselines: unordered bag of words models, which enable comparison to a standard NLP baseline, and information retrieval models, which allow us to compare against traditional question answering techniques.

BOW The BOW baseline is a logistic regression classifier trained on binary unigram indicators.[11] This simple discriminative model is an improvement over the generative quiz bowl answering model of Boyd-Graber et al. (2012).

BOW-DT The BOW-DT baseline is identical to BOW except we augment the feature set with dependency relation indicators. We include this baseline to isolate the effects of the dependency tree structure from our compositional model.

IR-QB The IR-QB baseline maps questions to answers using the state-of-the-art Whoosh IR engine. The knowledge base for IR-QB consists of "pages" associated with each answer, where each page is the union of training question text for that answer. Given a partial question, the text is first preprocessed using a query language similar to that of Apache Lucene. This processed query is then matched to pages uses BM-25 term weighting, and the top-ranked page is considered to be the model's guess. We also incorporate fuzzy queries to catch misspellings and plurals and use Whoosh's built-in query expansion functionality to add related keywords to our queries.

IR-WIKI The IR-WIKI model is identical to the IR-QB model except that each "page" in

[11] Raw word counts, frequencies, and TF-IDF weighted features did not increase performance, nor did adding bigrams to the feature set (possibly because multi-word named entities are already collapsed into single words).

its knowledge base also includes all text from the associated answer's Wikipedia article. Since all other baselines and **DTreeNN** models operate only on the question text, this is not a valid comparison, but we offer it to show that we can improve even this strong model using QANTA.

3.1.5 **DTreeNN** Configurations

For all **DTreeNN** models the vector dimension d and the number of wrong answers per node j is set to 100. All model parameters other than L are randomly initialized. The nonlinearity f is again the normalized $\tanh$ function.[12]

QANTA is our **DTreeNN** model with feature averaging across previously-seen sentences in a question. To obtain the final answer prediction given a partial question, we first generate a feature representation for each sentence within that partial question. This representation is computed by concatenating together the word embeddings and hidden representations averaged over all nodes in the tree as well as the root node's hidden vector. Finally, we send the average of all of the individual sentence features[13] as input to a logistic regression classifier for answer prediction.

FIXED-QANTA uses the same **DTreeNN** configuration as QANTA except the answer vectors are kept constant as in the text-to-image model.

[12]The standard $\tanh$ function produced heavy saturation at higher levels of the trees, and corrective weighting as in Socher et al. (2014) hurt our model because named entities that occur as leaves are often more important than non-terminal phrases.

[13]Initial experiments with L_2 regularization hurt performance on a validation set.

	History			Literature		
Model	Pos 1	Pos 2	Full	Pos 1	Pos 2	Full
BOW	27.5	51.3	53.1	19.3	43.4	46.7
BOW-DT	35.4	57.7	60.2	24.4	51.8	55.7
IR-QB	37.5	65.9	71.4	27.4	54.0	61.9
FIXED-QANTA	38.3	64.4	66.2	28.9	57.7	62.3
QANTA	**47.1**	**72.1**	**73.7**	**36.4**	**68.2**	**69.1**
IR-WIKI	53.7	76.6	77.5	41.8	74.0	73.3
QANTA+IR-WIKI	**59.8**	**81.8**	**82.3**	**44.7**	**78.7**	**76.6**

Table 3.1: Accuracy for history and literature at the first two sentence positions of each question and the full question. The top half of the table compares models trained on questions only, while the IR models in the bottom half have access to Wikipedia. QANTA outperforms all baselines that are restricted to just the question data, and it substantially improves an IR model with access to Wikipedia despite being trained on much less data.

3.1.6 Human Comparison

Previous work provides human answers (Boyd-Graber et al., 2012) for quiz bowl questions. We use human records for 1,201 history guesses and 1,715 literature guesses from twenty-two of the quiz bowl players who answered the most questions.[14]

The standard scoring system for quiz bowl is **10** points for a correct guess and **-5** points for an incorrect guess. We use this metric to compute a total score for each human. To obtain the corresponding score for our model, we force it to imitate each human's guessing policy. For example, Figure 3.1.6 shows a human answering in the middle of the second sentence. Since our model only considers sentence-level increments, we compare the model's prediction after the first sentence to the human prediction, which means our model is privy to less information than humans.

[14]Participants were skilled quiz bowl players and are not representative of the general population.

A minor character in this play can be summoned by a bell that does not always work; that character also doesn't have eyelids. Near the end, a woman who drowned her illegitimate child attempts to stab another woman in the Second Empire-style ◇ room in which the entire play takes place. For 10 points, Estelle and Ines are characters in which existentialist play in which Garcin claims "Hell is other people", written by Jean-Paul Sartre?

Figure 3.2: A question on the play "No Exit" with human buzz position marked as ◇. Since the buzz occurs in the middle of the second sentence, our model is only allowed to see the first sentence.

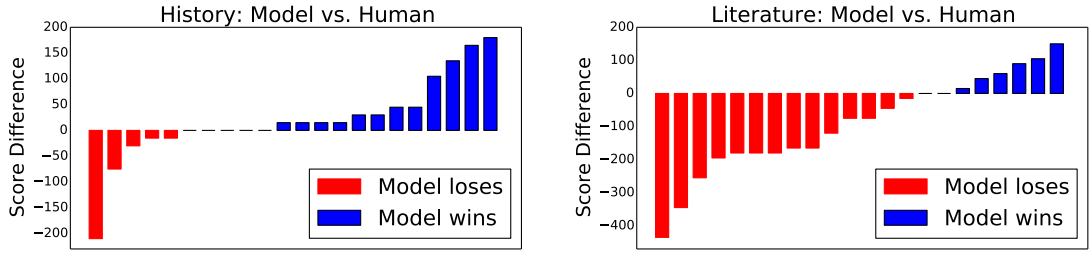


Figure 3.3: Comparisons of QANTA+IR-WIKI to human quiz bowl players. Each bar represents an individual human, and the bar height corresponds to the difference between the model score and the human score. Bars are ordered by human skill. Red bars indicate that the human is winning, while blue bars indicate that the model is winning. QANTA+IR-WIKI outperforms most humans on history questions but fails to defeat the "average" human on literature questions.

The resulting distributions are shown in Figure 3.1.6—our model does better than the average player on history questions, tying or defeating sixteen of the twenty-two players, but it does worse on literature questions, where it only ties or defeats eight players. The figure indicates that literature questions are harder than history questions for our model, which is corroborated by the experimental results discussed in the next section.

3.1.7 Discussion

In this section, we examine why QANTA improves over our baselines by giving examples of questions that are incorrectly classified by all baselines but correctly classified by QANTA. We also take a close look at some sentences that all models fail to answer correctly. Finally,

we visualize the answer space learned by QANTA.

3.1.7.1 Experimental Results

Table 3.1.5 shows that when bag of words and information retrieval methods are restricted to question data, they perform significantly worse than QANTA on early sentence positions. The performance of BOW-DT indicates that while the dependency tree structure helps by itself, the compositional distributed representations learned by QANTA are more useful. The significant improvement when we train answers as part of our vocabulary (see Section 3.1.2) indicates that our model uses answer occurrences within question text to learn a more informative vector space.

The disparity between IR-QB and IR-WIKI indicates that the information retrieval models need lots of external data to work well at all sentence positions. IR-WIKI performs better than other models because Wikipedia contains many more sentences that partially match specific words or phrases found in early clues than the question training set. In particular, it is impossible for all other models to answer clues in the test set that have no semantically similar or equivalent analogues in the training question data. With that said, IR methods can also operate over data that does not follow the special constraints of quiz bowl questions (e.g., every sentence uniquely identifies the answer, answers don't appear in their corresponding questions), which QANTA cannot handle. By combining QANTA and IR-WIKI, we are able to leverage access to huge knowledge bases along with deep compositional representations, giving us the best of both worlds.

3.1.7.2 Where the Attribute Space Helps Answer Questions

We look closely at the first sentence from a literature question about the author Thomas Mann: "He left unfinished a novel whose title character forges his father's signature to get out of school and avoids the draft by feigning desire to join".

All baselines, including IR-WIKI, are unable to predict the correct answer given only this sentence. However, QANTA makes the correct prediction. The sentence contains no named entities, which makes it almost impossible for bag of words or string matching algorithms to predict correctly. Figure 3.1.7.4 shows that the plot description associated with the "novel" node is strongly indicative of the answer. The five highest-scored answers are all male authors,[15] which shows that our model is able to learn the answer type without any hand-crafted rules.

Our next example, the first sentence in Table 3.1.7.4, is from the first position of a question on John Quincy Adams, which is correctly answered by only QANTA. The bag of words model guesses Henry Clay, who was also a Secretary of State in the nineteenth century and helped John Quincy Adams get elected to the presidency in a "corrupt bargain". However, the model can reason that while Henry Clay was active at the same time and involved in the same political problems of the era, he did not represent the Amistad slaves, nor did he negotiate the Treaty of Ghent.

[15] three of whom who also have well-known unfinished novels

3.1.7.3 Where all Models Struggle

Quiz bowl questions are intentionally written to make players work to get the answer, especially at early sentence positions. Our model fails to answer correctly more than half the time after hearing only the first sentence. We examine some examples to see if there are any patterns to what makes a question "hard" for machine learning models.

Consider this question about the Italian explorer John Cabot: "As a young man, this native of Genoa disguised himself as a Muslim to make a pilgrimage to Mecca".

While it is obvious to human readers that the man described in this sentence is not actually a Muslim, QANTA has to accurately model the verb *disguised* to make that inference. We show the score plot of this sentence in Figure 3.1.7.4. The model, after presumably seeing many instances of *muslim* and *mecca* associated with Mughal emperors, is unable to prevent this information from propagating up to the root node. On the bright side, our model is able to learn that the question is expecting a human answer rather than non-human entities like the Umayyad Caliphate.

More examples of impressive answers by QANTA as well as incorrect guesses by all systems are shown in Table 3.1.7.4.

3.1.7.4 Examining the Attribute Space

Figure 3.1.7.4 shows a t-SNE visualization (Van der Maaten and Hinton, 2008) of the 451 answers in our history dataset. The vector space is divided into six general clusters, and we focus in particular on the US presidents. Zooming in on this section reveals temporal clustering: presidents who were in office during the same timeframe occur closer together.

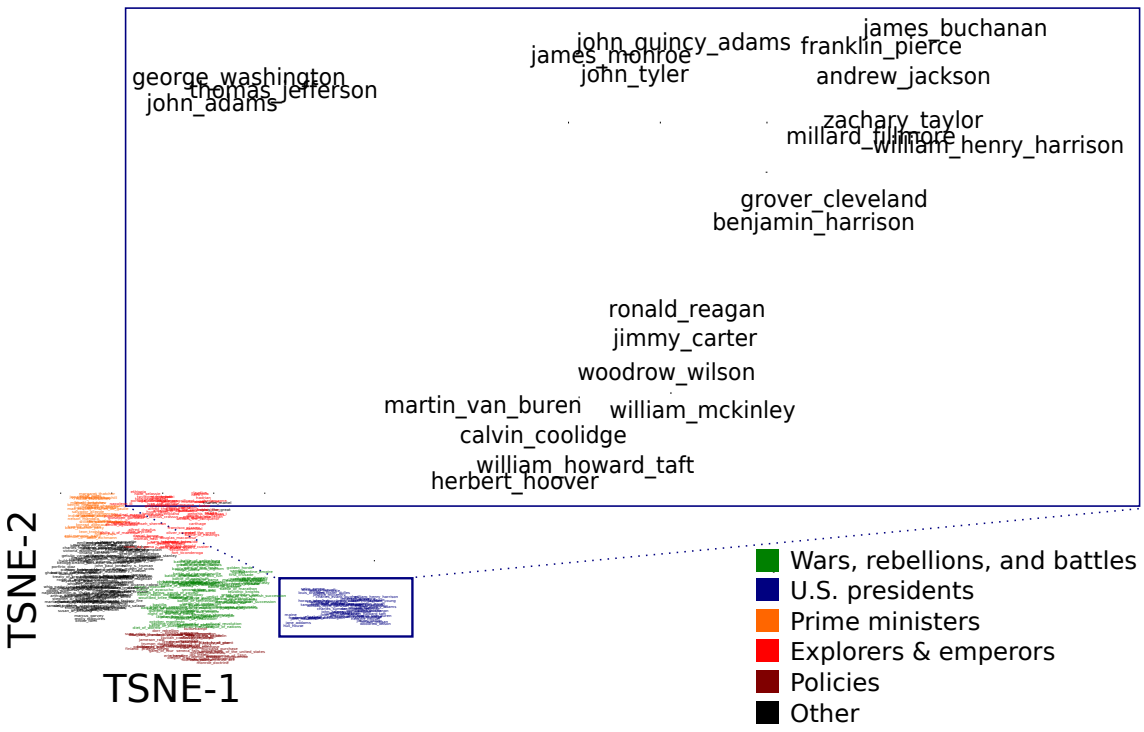


Figure 3.4: t-SNE 2-D projections of 451 answer vectors divided into six major clusters. The blue cluster is predominantly populated by U.S. presidents. The zoomed plot reveals temporal clustering among the presidents based on the years they spent in office.

This observation shows that QANTA is capable of learning attributes of entities during training.

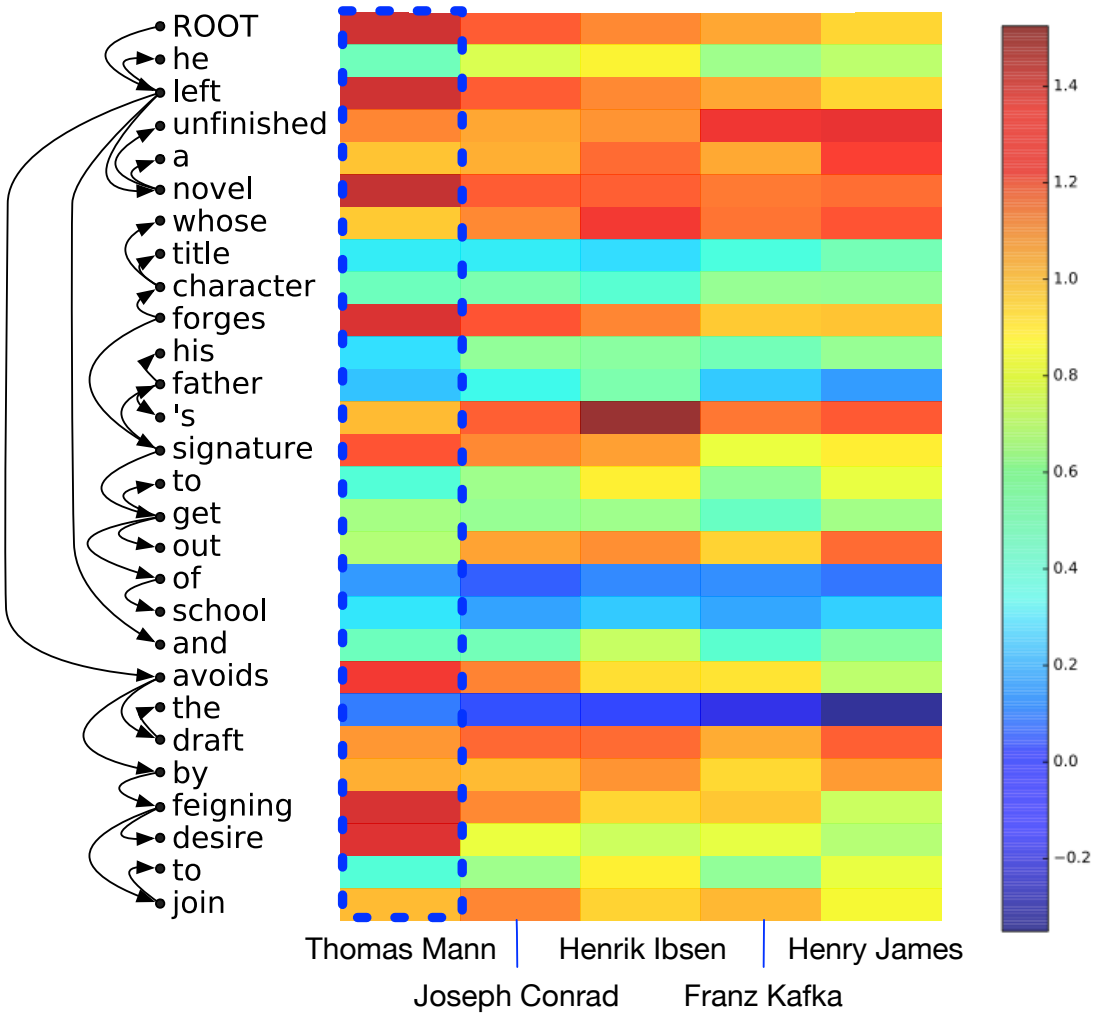


Figure 3.5: A question on the German novelist Thomas Mann that contains no named entities, along with the five top answers as scored by QANTA. Each cell in the heatmap corresponds to the score (inner product) between a node in the parse tree and the given answer, and the dependency parse of the sentence is shown on the left. All of our baselines, including IR-WIKI, are wrong, while QANTA uses the plot description to make a correct guess.

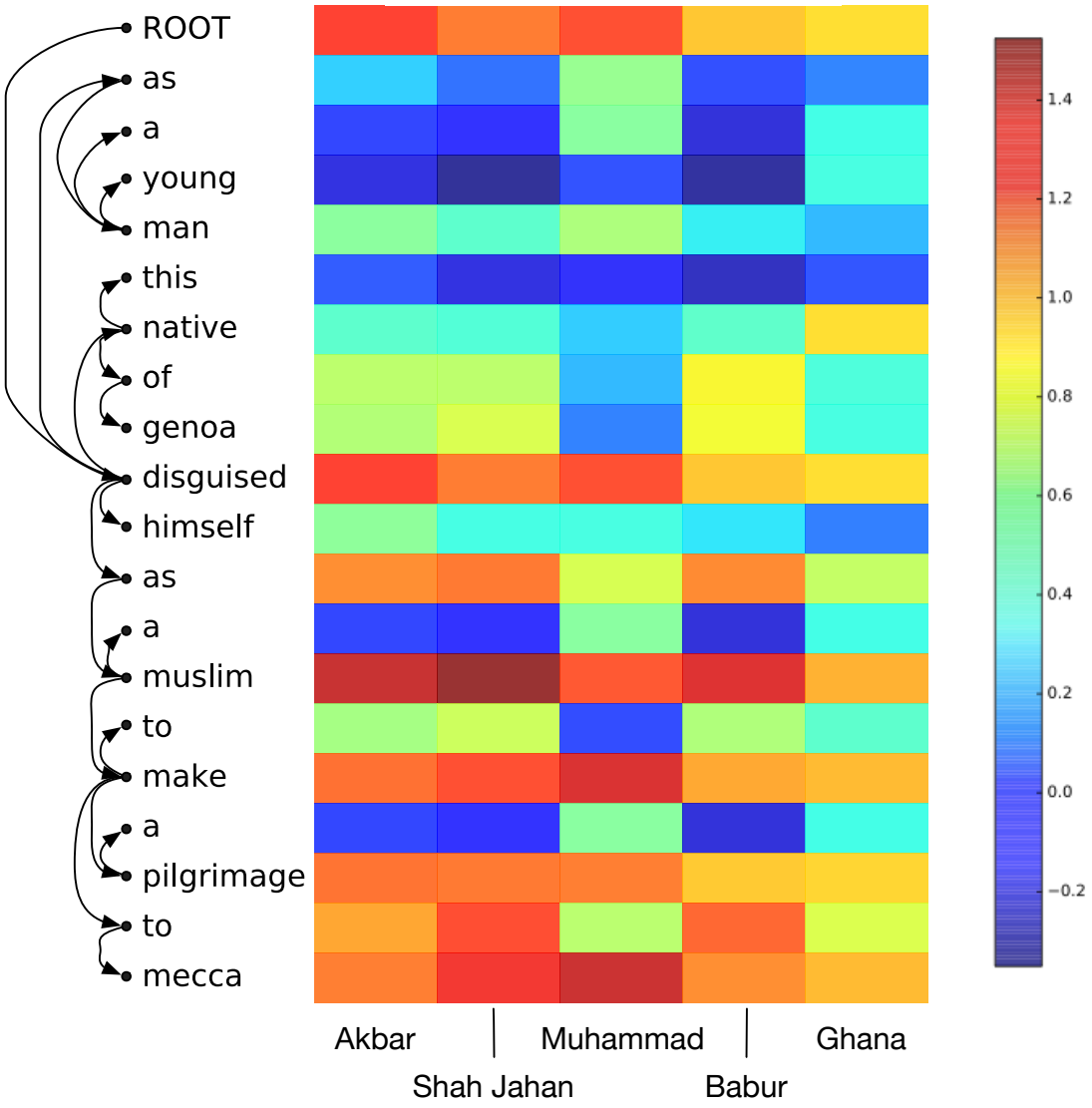


Figure 3.6: An extremely misleading question about John Cabot, at least to computer models. The words *muslim* and *mecca* lead to three Mughal emperors in the top five guesses from QANTA; other models are similarly led awry.

Q	he also successfully represented the amistad slaves and negotiated the treaty of ghent and the annexation of florida from spain during his stint as secretary of state under james monroe
A	**john quincy adams**, henry clay, andrew jackson
Q	this work refers to people who fell on their knees in hopeless cathedrals and who jumped off the brooklyn bridge
A	**howl**, the tempest, paradise lost
Q	despite the fact that twenty six martyrs were crucified here in the late sixteenth century it remained the center of christianity in its country
A	**nagasaki**, guadalcanal, ethiopia
Q	this novel parodies freudianism in a chapter about the protagonist 's dream of holding a live fish in his hands
A	billy budd, the ambassadors, all my sons
Q	a contemporary of elizabeth i he came to power two years before her and died two years later
A	grover cleveland, benjamin harrison, henry cabot lodge

Table 3.2: Five example sentences occuring at the first sentence position along with their top three answers as scored by QANTA; correct answers are marked with blue and wrong answers are marked with red. QANTA gets the first three correct, unlike all other baselines. The last two questions are too difficult for all of our models, requiring external knowledge (e.g., *Freudianism*) and temporal reasoning.

3.2 Simpler Quiz Bowl Models

QANTA is a relatively complex model, containing many different composition matrices and relying on sentential parse trees for the composition order. After finishing the QANTA project, I wanted to explore different neural network architectures for quiz bowl. During experimentation, a simple word vector average yielded highly competitive results, despite the fact that it throws out all word order information. The deep averaging network, described in Chapter 2, was borne out of these experiments. To conclude this chapter, I revisit the quiz bowl task with a **DAN** instead of **DTreeNN** and discover that averaging is an effective sentence composition method when paired with a novel dropout variant; this result influences design decisions made for the **RMN** model described in the next chapter.

Model	Pos 1	Pos 2	Full	Time(s)
BoW-DT	35.4	57.7	60.2	—
IR	37.5	65.9	71.4	N/A
QANTA	47.1	72.1	73.7	314
DAN	46.4	70.8	71.8	**18**
IR-WIKI	53.7	**76.6**	**77.5**	N/A
QANTA-WIKI	46.5	72.8	73.9	1,648
DAN-WIKI	**54.8**	75.5	77.1	119

Table 3.3: The **DAN** achieves slightly lower accuracies than the more complex **QANTA** in much less training time, even at early sentence positions where compositionality plays a bigger role. When Wikipedia is added to the training set (bottom half of table), the **DAN** outperforms **QANTA** and achieves comparable accuracy to a state-of-the-art information retrieval baseline, which highlights a benefit of ignoring word order for this task.

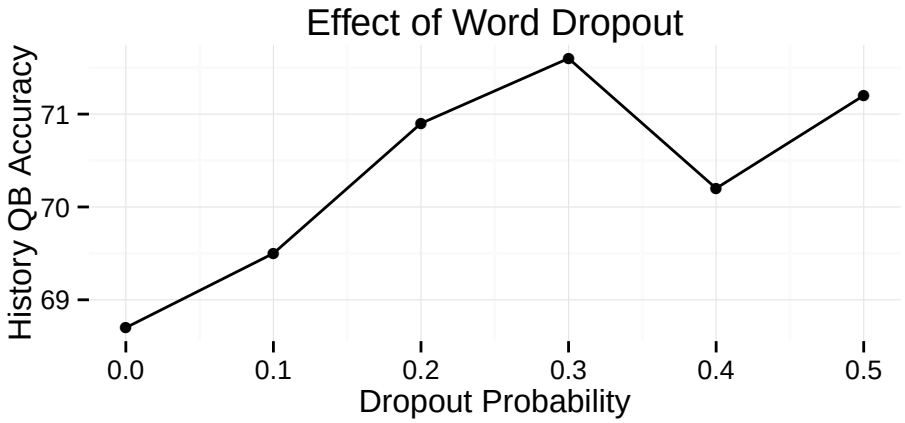


Figure 3.7: Randomly dropping out 30% of words from the vector average is optimal for the quiz bowl task, yielding a gain in absolute accuracy of almost 3% on the quiz bowl question dataset compared to the same model trained with no word dropout.

3.2.1 **DAN**s for Quiz Bowl

On the same quiz bowl dataset, the **DAN** outperforms other bag-of-words models and is competitive with QANTA, but requires much less training time. More interestingly, we

find that unlike the **TreeNN**, the **DAN** significantly benefits from out-of-domain Wikipedia training data. As a reminder, QANTA's dependency-tree **TreeNN**shows substantial improvements over other **QA** methods, leading to the hypothesis that correctly modeling compositionality is crucial for answering hard questions. Before describing the experiments, I introduce *word dropout*, a regularization method that can benefit any neural composition function.

3.2.1.1 Word Dropout Improves Robustness

Dropout regularizes neural networks by randomly setting hidden and/or input units to zero with some probability p (Hinton et al., 2012; Srivastava et al., 2014). Given a neural network with n units, dropout prevents overfitting by creating an ensemble of 2^n different networks that share parameters, where each network consists of some combination of dropped and undropped units. Instead of dropping units, a natural extension for the **DAN** model is to randomly drop word tokens' entire *word embeddings* from the vector average. Another way to view word dropout is as an application of dropout to the one-hot encoding (Section 2.1) of a given sentence. Using this method, which we call *word dropout*, our network theoretically sees $2^{|X|}$ different token sequences for each input X.

We posit a vector $\boldsymbol{r}$ with $|X|$ independent Bernoulli trials, each of which equals 1 with probability p. The embedding $\boldsymbol{v}_w$ for token w in X is dropped from the average if $\boldsymbol{r}_w$ is 0, which exponentially increases the number of unique examples the network sees

during training. This allows us to modify Equation 2.1:

$$\boldsymbol{r}_w \sim \text{Bernoulli}(p) \tag{3.4}$$

$$\hat{X} = \{w | w \in X \text{ and } \boldsymbol{r}_w > 0\} \tag{3.5}$$

$$\boldsymbol{z} = g(w \in X) = \frac{\sum_{w \in \hat{X}} \boldsymbol{v}_w}{|\hat{X}|}. \tag{3.6}$$

Depending on the choice of p, many of the "dropped" versions of an original training instance will be very similar to each other, but for shorter inputs this is less likely. We might drop a very important token, such as "horrible" in "the crab rangoon was especially horrible"; however, since the number of word types that are predictive of the output labels is low compared to non-predictive ones (e.g., neutral words in sentiment analysis), we always see improvements using this technique.

Theoretically, word dropout can also be applied to other neural network-based approaches. However, we observe no significant performance differences in preliminary experiments when applying word dropout to leaf nodes in **TreeNN**s for sentiment analysis (dropped leaf representations are set to zero vectors), and it slightly hurts performance on the question answering task.

Dataset and Experimental Setup We train a **DAN** over the history questions from Iyyer et al. (2014a).[16] This dataset is augmented with 49,581 sentence/page-title pairs from the Wikipedia articles associated with the answers in the dataset. For fair comparison with QANTA, we use a normalized tanh activation function at the last layer instead of ReLu,

[16]The training set contains 14,219 sentences over 3,761 questions. For more detail about data and baseline systems, see Iyyer et al. (2014a).

and we also change the output layer from a softmax to the margin ranking loss Weston et al. (2011) used in **QANTA**. We initialize the **DAN** with the same pretrained 100-d word embeddings that were used to initialize QANTA.

We also evaluate the effectiveness of word dropout on this task in Figure 3.2. Cross-validation indicates that $p = 0.3$ works best for question answering, although the improvement in accuracy is negligible for sentiment analysis. Finally, continuing the trend observed in the sentiment experiments, **DAN** converges much faster than QANTA.

DANs Improve with Noisy Data Table 3.2 shows that while **DAN** is slightly worse than QANTA when trained only on question-answer pairs, it improves when trained on additional out-of-domain Wikipedia data (**DAN-WIKI**), reaching performance comparable to that of a state-of-the-art information retrieval system (**IR-WIKI**). **QANTA**, in contrast, barely improves when Wikipedia data is added (**QANTA-WIKI**) possibly due to the syntactic differences between Wikipedia text and quiz bowl question text.

The most common syntactic structures in quiz bowl sentences are imperative constructions such as "Identify this British author who wrote Wuthering Heights", which are almost never seen in Wikipedia. Furthermore, the subject of most quiz bowl sentences is a pronoun or pronomial mention referring to the answer, a property that is not true of Wikipedia sentences (e.g., "Little of Emily's work from this period survives, except for poems spoken by characters."). Finally, many Wikipedia sentences do not uniquely identify the title of the page they come from, such as the following sentence from Emily Brontë's page: "She does not seem to have made any friends outside her family." While noisy data affect both **DAN** and QANTA, the latter is further hampered by the syntactic

divergence between quiz bowl questions and Wikipedia, which may explain the lack of improvement in accuracy.

3.3 Conclusion

In this chapter, I first introduce QANTA, a dependency-tree neural network for factoid question answering that outperforms bag of words and information retrieval baselines. The model improves upon a contrastive max-margin objective function from previous work to dynamically update answer vectors during training with a single model. Additionally, I show that sentence-level representations can be easily and effectively combined to generate paragraph-level representations with more predictive power than those of the individual sentences. In the final section of the chapter, I show that the complex machinery of the **DTreeNN** is actually unnecessary for quiz bowl, and that deep bag-of-words models can achieve similar performance with much less training time.

Chapter 4

Sequential Semantic Parsing: Integrating Context with Structured Prediction

In the previous chapter, we formalize quiz bowl as a classification problem and used very simple methods to extract information from discourse-level context. *Semantic parsing*, which is the task I explore in this chapter, cannot be tackled in the same way. A semantic parser maps natural language text to meaning representations in formal logic (Liang, 2016); instead of producing an answer prediction as in classification, the goal here is to produce a formal query (or semantic parse) of the question. Once a natural language question has been mapped to a formal query, its answer can be retrieved by executing the query on a back-end structured database. Semantic parsing is well-studied for single-sentence questions, but how can it be extended to cover conversational QA with dependencies between turns?

This chapter includes content and figures previously published in Iyyer et al. (2017b), the work for which was done during an internship at Microsoft Research in 2016.

4.1 Motivating Contextual Understanding for Semantic Parsing

One of the main focuses of semantic parsing research is how to address *compositionality* in language, and complicated questions have been specifically targeted in the design of a recently-released QA dataset (Pasupat and Liang, 2015). Take for example the following

question: "*of those actresses who won a Tony after 1960, which one took the most amount of years after winning the Tony to win an Oscar?*" The corresponding logical form is highly compositional; in order to answer it, many sub-questions must be implicitly answered in the process (e.g., *"who won a Tony after 1960?"*).

While semantic parsers *should* be able to answer very complicated questions, in reality these questions are rarely issued by users.[1] Because users can interact with a QA system repeatedly, there is no need to assume a single-turn QA setting where the exact *question intent* has to be captured with just one complex question. The same intent can be more naturally expressed through a sequence of simpler questions, as shown below:

1. *What actresses won a Tony after 1960?*
2. *Of those, who later won an Oscar?*
3. *Who had the biggest gap between their two award wins?*

Decomposing complicated intents into multiple related but simpler questions is arguably a more effective strategy to explore a topic of interest, and it reduces the cognitive burden on both the person who asks the question and the one who answers it.[2]

In this work, we study semantic parsing for answering *sequences* of simple related questions. We collect a dataset of question sequences called SequentialQA (SQA; Section 4.2)[3] by asking crowdsourced workers to decompose complicated questions sampled from the WikiTableQuestions dataset (Pasupat and Liang, 2015) into multiple easier ones. SQA, which contains 6,066 question sequences with 17,553 total question-answer

[1] For instance, there are only 3.75% questions with more than 15 words in WikiAnswers (Fader et al., 2014).

[2] Studies have shown increased sentence complexity links to longer reading times (Hale, 2006; Levy, 2008; Frank, 2013).

[3] Available at http://aka.ms/sqa

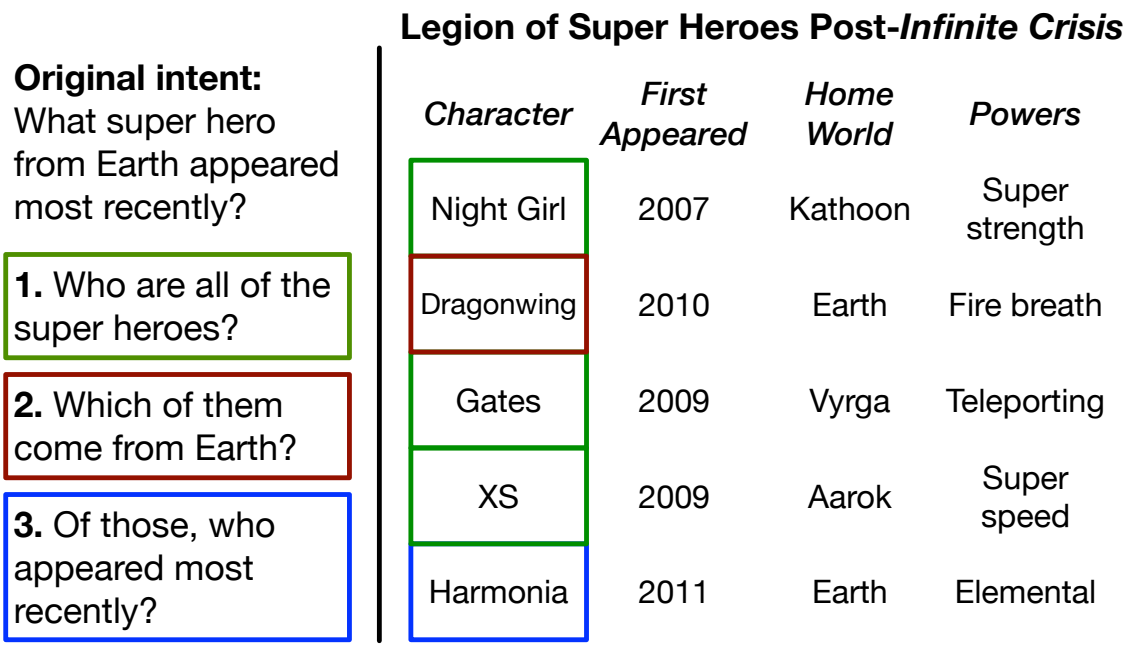


Figure 4.1: An example question sequence created from a compositional question intent. Workers must write questions whose answers are subsets of cells in the table.

pairs, is to the best of our knowledge the first semantic parsing dataset for sequential question answering. Section 4.3 describes our novel dynamic neural semantic parsing framework (`DynSP`), a weakly supervised structured-output learning approach based on reward-guided search that is designed for solving sequential QA. We demonstrate in Section 4.4 that `DynSP` achieves higher accuracies than existing systems on SQA, and we offer a qualitative analysis of question types that our method answers effectively, as well as those on which it struggles.

4.2 A Dataset of Question Sequences

We collect the SequentialQA (SQA) dataset via crowdsourcing by leveraging WikiTableQuestions (Pasupat and Liang, 2015, henceforth WTQ), which contains highly composi-

tional questions associated with HTML tables from Wikipedia.

Each crowdsourcing task contains a long, complex question originally from WTQ as the question *intent*. The workers are asked to compose a sequence of simpler questions that lead to the final intent; an example of this process is shown in Figure 4.2.

To simplify the task for workers, we only use questions from WTQ whose answers are cells in the table, which excludes those involving arithmetic and counting. We likewise also restrict the questions our workers can write to those answerable by only table cells. These restrictions speed the annotation process because workers can just click on the table to answer their question. They also allow us to collect answer coordinates (row and column in the table) as opposed to answer text, which removes many normalization issues for answer string matching in evaluation. Finally, we only use long questions that contain nine or more words as intents; shorter questions tend to be simpler and are thus less amenable to decomposition.

4.2.1 Properties of SQA

In total, we use 2,022 question intents from the train and test folds of the WTQ for decomposition. Three workers decompose each intent, resulting in 6,066 unique questions sequences containing 17,553 total question-answer pairs (for an average of 2.9 questions per sequence). We divide the dataset into train and test using the original WTQ folds, resulting in an 83/17 train/test split. Importantly, just like in WTQ, none of the tables in the test set are in the training set.

We identify three frequently-occurring question classes: *column selection*, *subset*

selection, and *row selection*.[4] In column selection questions, the answer is an entire column of the table; these questions account for 23% of all questions in SQA. Subset and row selection are more complicated than column selection, as they usually contain coreferences to the previous question's answer. In subset selections, the answer is a subset of the previous question's answer; similarly, the answers to row selections occur in the same row(s) as the previous answer but in a different column. Subset selections make up 27% of SQA, while row selections are an additional 19%. The remaining 31% contains more complex combinations of these three types.

We also observe dramatic differences in the types of questions that are asked at each position of the sequence. For example, 51% of the first questions in the sequences are column selections (e.g., "*what are all of the teams?*"). This number dwindles to just 18% when we look at the second question of each sequence, which indicates that the collected sequences start with general questions and progress to more specific ones.

4.3 Dynamic Neural Semantic Parsing

The unique setting of SQA provides both opportunities and challenges. On the one hand, it contains short questions with less compositionality, which in theory should reduce the difficulty of the semantic parsing problem; on the other hand, the additional contextual dependencies of the preceding questions and their answers increase modeling complexity. These observations lead us to propose a dynamic neural semantic parsing framework (`DynSP`) trained using a reward-guided search procedure for solving SQA.

[4] In the example sequence "*what are all of the tournaments? in which one did he score the least points? on what date was that?*", the first question is a column selection, the second is a subset selection, and the last one is a row selection.

Given a question (optionally along with previous questions and answers) and a table, DynSP formulates the semantic parsing problem as a state–action search problem. Each state represents a complete or partial parse, while each action corresponds to an operation to extend a parse. The goal during inference is to find an end state with the highest score as the predicted parse.

The quality of the induced semantic parse obviously depends on the scoring function. In our design, the score of a state is determined by the scores of actions taken from the initial state to the target state, which are predicted by different neural network modules based on action type. By leveraging a margin-based objective function, the model learning procedure resembles several structured-output learning algorithms such as structured SVMs (Tsochantaridis et al., 2005), but can take either strong or weak supervision seamlessly.

DynSP is inspired by STAGG, a search-based semantic parser (Yih et al., 2015), as well as the dynamic neural module network (DNMN) of Andreas et al. (2016). Much like STAGG, DynSP chains together different modules as search progresses; however, these modules are implemented as neural networks, which enables end-to-end training as in DNMN. The key difference between DynSP and DNMN is that in DynSP the network structure of an example is not predetermined. Instead, different network structures are constructed dynamically as our learning procedure explores the state space.

It is straightforward to answer *sequential* questions using our framework: we allow the model to take the previous question and its answers as input, with a slightly modified action space to reflect a *dependent* semantic parse. The same search and learning procedure is then able to effortlessly adapt to the new setting. In this section, we first describe the

formal language underlying DynSP, followed by the model formulation and learning algorithm.

4.3.1 Semantic parse language

Because tables are used as the data source to answer questions in SQA, we decide to form our semantic parses in an SQL-like language[5]. Our parses consist of two parts: a ***select*** statement and conjunctions of zero or more ***conditions***.

A ***select*** statement is associated with a column name, which is referred to as the *answer* column. Conditions enforce additional constraints on which cells in the answer column can be chosen; a ***select*** statement without any conditions indicates that an entire column of the table is the answer to the question. In particular, each condition contains a column name as the *condition* column and an operator with zero or more arguments. The operators in this work include: $=, \neq, >, \geq, <, \leq, \arg\min, \arg\max$. A cell in the answer column is only a legitimate answer if the cell of the corresponding row in the condition column satisfies the constraint defined by the operator and its arguments.

As a concrete example, suppose the data source is the same table in Fig. 4.2. The semantic parse of the question "*Which super heroes came from Earth and first appeared after 2009?*" is "Select **Character** Where {**Home World** = *Earth*} $\wedge$ {**First Appeared** $>$ *2009*}" and the answers are {Dragonwing, Harmonia}.

To handle the *sequential* aspect of SQA, we extend the semantic parse language by adding a preamble statement ***subsequent***. A ***subsequent*** statement contains only condi-

[5]Our framework is not restricted to the formal language we use in this work. In addition, the structured query can be straightforwardly represented in other formal languages, such as the lambda DCS logic used in (Pasupat and Liang, 2015).

tions, as it essentially adds constraints to the semantic parse of the previous question. For instance, if the follow-up question is "*Which of them breathes fire?*", then the corresponding semantic parse is "Subsequent Where {**Powers** = *Fire breath*}". The answer to this question is {Dragonwing}, a subset of the previous answer.

4.3.2 Model formulation

We start introducing our model design by first defining the state and action space. Let $\mathcal{S}$ be the set of states and $\mathcal{A}$ the set of all actions. A state $s \in \mathcal{S}$ is simply a sequence of variable length of actions $\{a_1, a_2, a_3, \cdots, a_t\}$, where $a_i \in \mathcal{A}$. An empty sequence, $s_0 = \phi$, is a special state used as the starting point of search.

As mentioned earlier, a state represents a (partial) semantic parse of *one* question. Each action is thus a legitimate operation that can be added to *grow* the semantic parse. Our action space design is tied closely to the statements defined by our parse language; in particular, an action *instance* is either a complete or partial statement, and action instances are grouped by *type*. For example, ***select*** and ***subsequent*** operations are two action types. A ***condition*** statement is formed by two different action types: (1) selection of the condition column, and (2) the comparison operator. The instances of each action type differ in their arguments (e.g., column names, or specific cells in a column). Because conditions in a ***subsequent*** parse rely on previous questions and answers, they belong to different action types from regular conditions. Table 4.3.2 summarizes the action space defined in this work.

Any state that represents a complete and legitimate parse is an end state. Search does

Id	Type	# Action instances
$\mathcal{A}_1$	Select-column	# columns
$\mathcal{A}_2$	Cond-column	# columns
$\mathcal{A}_3$	Op-Equal ($=$)	# rows
$\mathcal{A}_4$	Op-NotEqual ($\neq$)	# rows
$\mathcal{A}_5$	Op-GT ($>$)	# numbers / datetimes
$\mathcal{A}_6$	Op-GE ($\geq$)	# numbers / datetimes
$\mathcal{A}_7$	Op-LT ($<$)	# numbers / datetimes
$\mathcal{A}_8$	Op-LE ($\leq$)	# numbers / datetimes
$\mathcal{A}_9$	Op-ArgMin	# numbers / datetimes
$\mathcal{A}_{10}$	Op-ArgMax	# numbers / datetimes
$\mathcal{A}_{11}$	Subsequent	1
$\mathcal{A}_{12}$	S-Cond-column	# columns
$\mathcal{A}_{13}$	S-Op-Equal ($=$)	# rows
$\mathcal{A}_{14}$	S-Op-NotEqual ($\neq$)	# rows
$\mathcal{A}_{15}$	S-Op-GT ($>$)	# numbers / datetimes
$\mathcal{A}_{16}$	S-Op-GE ($\geq$)	# numbers / datetimes
$\mathcal{A}_{17}$	S-Op-LT ($<$)	# numbers / datetimes
$\mathcal{A}_{18}$	S-Op-LE ($\leq$)	# numbers / datetimes
$\mathcal{A}_{19}$	S-Op-ArgMin	# numbers / datetimes
$\mathcal{A}_{20}$	S-Op-ArgMax	# numbers / datetimes

Table 4.1: Types of actions and the number of action instances in each type. Numbers / datetimes are the mentions discovered in the question (plus the previous question if it is a subsequent condition).

not necessarily need to stop at an end state, because adding more actions (e.g., condition statements) can lead to another end state. Take the same example question from before: "*Which super heroes came from Earth and first appeared after 2009?*". One action sequence that represents the parse is $\{(\mathcal{A}_1)$ select-column **Character**, $(\mathcal{A}_2)$ cond-column **Home World**, $(\mathcal{A}_3)$ op-equal *Earth*, $(\mathcal{A}_2)$ cond-column **First Appeared**, $(\mathcal{A}_5)$ op-gt *2009*$\}$.

Many states represent semantically equivalent parses (e.g., those with the same actions ordered differently, or states with repeated conditions). To prune the search space, we introduce the function $Act(s) \subset \mathcal{A}$, which defines the actions that can be taken when given a state s. Borrowing the idea of *staged* state generation in Yih et al. (2015), we choose a default ordering of actions based on their types, dictating that a *select* action must

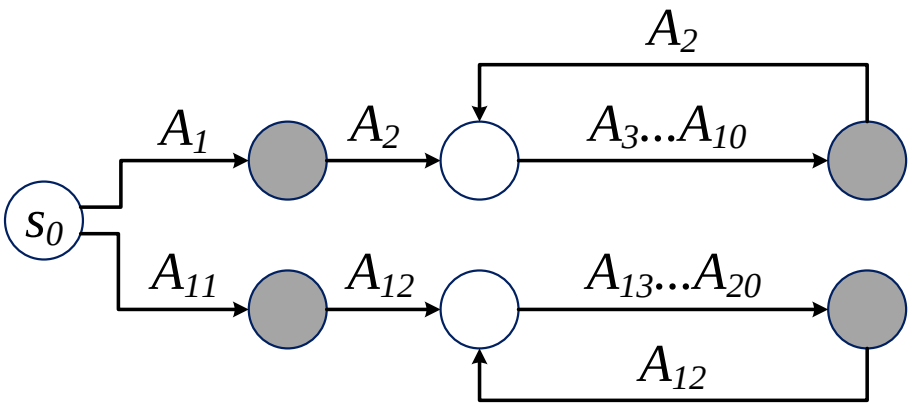


Figure 4.2: Possible action transitions based on their types (see Table 4.3.2). Shaded circles are end states.

be picked first and that a *condition-column* needs to be determined before the *operator* is chosen. The full transition diagram is presented in Fig. 4.3.2. To implement this transition order, we only need to check the last action in the state. In addition, we also disallow adding duplicates of actions that already exist in the state.

We use beam search to find an end state with the highest score for inference. Let s_t be a state consisting of a sequence of actions $a_1, a_2, \cdots, a_t$. The state value function V is defined recursively as $V(s_t) = V(s_{t-1}) + \pi(s_{t-1}, a_t), V(s_0) = 0$, where the *policy* function $\pi(s, a)$ scores an action $a \in Act(s)$ given the current state.

4.3.3 Policy function

The intuition behind the policy function can be summarized as follows. Halfway through the construction of a semantic parse, the policy function measures the quality of an immediate action that can be taken next given the current state (i.e., the question and actions that have previously been chosen). To enable integrated, end-to-end learning, the policy function in our framework is parameterized using neural networks. Because

each action type has very different semantics, we design different network structures (i.e., modules) accordingly.

Most of our network structures encourage learning semantic matching functions between the words in the question and table (either the column names or cells). Here we illustrate the design using the *select-column* action type ($\mathcal{A}_1$). Conceptually, the corresponding module is a combination of various matching scores. Let W_Q be the embeddings of words in the question and W_C be the embeddings of words in the target column name. The component matching functions are:

$$
\begin{aligned}
f_{max} &= \frac{1}{|W_C|} \sum_{w_c \in W_C} \max_{w_q \in W_Q} w_q^T w_c \\
f_{avg} &= \left(\frac{1}{|W_C|} \sum_{w_c \in W_C} w_c \right)^T \left(\frac{1}{|W_Q|} \sum_{w_q \in W_Q} w_q \right)
\end{aligned}
$$

Essentially, for each word in the column name, f_{max} finds the highest matching question word and outputs the average score. Conversely, f_{avg} simply uses the average word vectors of the question and column name and returns their inner product. In another variant of f_{avg}, we replace the question representation with the output of a bi-directional LSTM model. These matching component functions are combined by a 2-layer feed-forward neural network, which outputs a scalar value as the action score.

4.3.4 Model learning

Because the state value function V is defined recursively as the sum of scores of actions in the sequence, the goal of model optimization is to learn the parameters in the neural networks behind the policy function. Let θ be the collection of all the model parameters. Then the state value function can be written as: $V_\theta(s_t) = \sum_{i=1}^{t} \pi_\theta(s_{i-1}, a_i)$.

In a fully supervised setting where the correct semantic parse of each question is available, learning the policy function can be reduced to a sequence prediction problem. However, while having full supervision leads to a better semantic parser, collecting the correct parses requires a much more sophisticated UI design (Yih et al., 2016). In many scenarios, such as the one in the SQA dataset, it is often the case that only the answers to the questions are available. Adapting a learning algorithm to this *weakly* supervised setting is thus critical.

Generally speaking, weakly supervised semantic parsers operate on one assumption—a candidate semantic parse is treated as a correct one if it results in answers that are identical to the gold answers. Therefore, a straightforward modification of existing structured learning algorithms in our setting is to use any semantic parse found to evaluate to the correct answers during beam search as a *reference* parse, and then update the model parameters accordingly. In practice, however, this approach is often problematic: the search space can grow enormously, and when coupled with poor model performance early during training, this leads to beams that contain no parses evaluating to the correct answer. As a result, learning becomes inefficient and takes a long time to converge.

In this work, we propose a conceptually simple learning algorithm for weakly

supervised training that sidesteps the inefficient learning problem. Our key insight is to conduct inference using a beam search procedure guided by an *approximate reward* function. The search procedure is executed *twice* for each training example, one for finding the best possible reference semantic parse and the other for finding the predicted semantic parse to update the model. Our framework is suitable for learning from either implicit or explicit supervision. Below we describe how we adapt it to the semantic parsing problem in this work.

Approximate reward Let $A(s)$ be the answers retrieved by executing the semantic parse represented by state s, and let A^* be the set of gold answers of a given question. We define the *reward* $R(s; A^*) = \mathbb{1}[A(s) = A^*]$, or the accuracy of the retrieved answers. We use $R(s)$ as the abbreviation for $R(s; A^*)$. A state s with $R(s) = 1$ is called a goal state. Directly using this reward function in search of goal states can be difficult, as rewards of most states are 0. However, even when the answers from a semantic parse are not completely correct, some overlap with the gold answers can still hint that the state is close to a goal state, thus providing useful information to guide search. To formalize this idea, we define an *approximated reward* $\tilde{R}(s)$ in this work using the Jaccard coefficient ($\tilde{R}(s) = |A(s) \cap A^*|/|A(s) \cup A^*|$). If s is a goal state, then obviously $\tilde{R}(s) = R(s) = 1$. Also because our actions effectively add additional constraints to exclude some table cells, any succeeding states of s' with $\tilde{R}(s') = 0$ will also have 0 approximate reward and can be pruned from search immediately.

We use the approximate reward $\tilde{R}$ to guide our beam search to find the reference parses (i.e., goal states). Some variations of the approximate reward can make learning

more efficient. For instance, we use the model score for tie-breaking, effectively making the approximate reward function depend on the model parameters:

$$\tilde{R}_\theta(s) = |A(s) \cap A^*|/|A(s) \cup A^*| + \epsilon V_\theta(s), \tag{4.1}$$

where ϵ is a small constant. When a goal state is not found, the state with the highest approximate reward can still be used as a surrogate reference.

Updating parameters The model parameters are updated by first finding the most *violated* state $\hat{s}$ and then comparing $\hat{s}$ with a reference state s^* to compute a loss. The idea of finding the most violated state comes from Taskar et al. (2004), with the intuition that the learning algorithm should make the state value function behave similarly to the reward. Formally, for every state s, we would like the value function to satisfy the following constraint:

$$V_\theta(s^*) - V_\theta(s) \geq R(s^*) - R(s) \tag{4.2}$$

$R(s^*) - R(s)$ is thus the margin. As discussed above, we use approximate reward function $\tilde{R}_\theta$ instead of the true reward. We want to update the model parameters θ to make sure that the constraint is satisfied. When the constraint is violated, the degree of violation can be written as:

$$\mathcal{L}(s) = V_\theta(s) - V_\theta(s^*) - \tilde{R}_\theta(s) + \tilde{R}_\theta(s^*) \tag{4.3}$$

In the algorithm, we want to find the state such that the corresponding constraint is most violated. Finding the most violated state is then equivalent to finding the state with the

Algorithm 1 Model parameter updates

1: **for** pick a labeled data (x, A^*) **do**
2: $s^* \leftarrow \arg\max_{s \in \mathcal{E}(x)} \tilde{R}(s; A^*)$
3: $\hat{s} \leftarrow \arg\max_{s \in \mathcal{E}(x)} V_\theta(s) - \tilde{R}(s; A^*)$
4: update θ by minimizing $\max(\mathcal{L}(s), 0)$
5: **end for**

highest value of $V_\theta(s) - \tilde{R}_\theta(s)$ as the other two terms are constant.

Algorithm 4.3.4 sketches the key steps of our method in each iteration. It first picks a training instance (x and y), where x represents the table and the question, and y is the gold answer set. The approximate reward function $\tilde{R}$ is defined by y, while $\mathcal{E}(x)$ is the set of end states for this instance. Line 2 finds the best reference and Line 3 finds the most violated state, both relying on beam search for approximate inference. Line 4 computes the gradient of the loss in Eq. Eq. (4.3), which is then used in backpropagation to update the model parameters.

4.4 Experiments

Since the questions in SQA are decomposed from those in WTQ, we compare our method, `DynSP`, to two existing semantic parsers designed for WTQ: (1) the *floating parser* (`FP`) of Pasupat and Liang (2015), and (2) the *neural programmer* (`NP`) of Neelakantan et al. (2017). We describe below each system's configurations in more detail and qualitatively compare and contrast their performance on SQA.

Floating parser: The floating parser (Pasupat and Liang, 2015) maps questions to logical forms and then executes them on the table to retrieve the answers. It was designed

specifically for the WTQ task (achieving 37.0% accuracy on the WTQ test set) and differs from other semantic parsers by not anchoring predicates to tokens in the question, relying instead on typing constraints to reduce the search space. Using `FP` as-is results in poor performance on SQA because the system is configured for questions with single answers, while SQA contains many questions with multiple-cell answers. We address this issue by removing a pruning hyperparameter (*tooManyValues*) and features that add bias on the denotation size.

Neural programmer: The neural programmer (`NP`) proposed by Neelakantan et al. (2017) has shown promising results on WTQ, achieving accuracies on par with those of `FP`. Similar to our method, `NP` contains specialized neural modules that perform discrete operations such as argmax and argmin, and it is able to chain together multiple modules to answer a single question. However, module selection in `NP` is computed via soft attention (Cho et al., 2014), and information is propagated from one module to the next using a recurrent neural network. Since module selection is not tied to a pre-defined parse language like `DynSP`, `NP` simply runs for a fixed number of recurrent timesteps per question rather than growing a parse until it is complete.

Comparing the baseline systems: `FP` and `NP` exemplify two very different paradigms for designing a semantic parsing system to answer questions using structured data. `FP` is a feature-rich system that aims to output the correct semantic parse (in a logical parse language) for a given question. On the other hand, the end-to-end neural network of `NP` relies on its modular architectures to output a probability distribution over cells in a table

given a question. While NP can learn more powerful neural matching functions between questions and tables than FP's simpler feature-based matching, NP cannot produce a complete, discrete semantic parse, which means that its actions can only be interpreted coarsely by looking at the order of the modules selected at each timestep.[6] Furthermore, FP's design theoretically allows it to operate on partial tables indirectly through an API, which is necessary if tables are large and stored in a backend database, while NP requires upfront access to the full tables to facilitate end-to-end model differentiability.[7]

Even though FP and NP are powerful systems designed for the more difficult, compositional questions in WTQ, our method outperforms both systems on SQA when we consider all questions within a sequence independently of each other (a fair comparison), demonstrating the power of our search-based semantic parsing framework. More interestingly, when we leverage the sequential information by including the ***subsequent*** action, our method improves almost 3% in absolute accuracy.

DynSP combines the best parts of both FP and NP. Given a question, we try to generate its correct semantic parse in a formal language that can be predefined by the choice of structured data source (e.g., SQL). However, we push the burden of feature engineering to neural networks as in NP. Our framework is easier to extend to the sequential setting of SQA than either baseline system, requiring just the additional ***subsequent*** action. FP's reliance on a hand-designed grammar necessitates extra rules that operate over partial tables from the previous question, which if added would blow up the search space. Meanwhile, modifying NP to handle sequential QA is non-trivial due to soft module and

[6] Since NP uses a fixed number of timesteps for each question, the module order is not guaranteed to correspond to a complete parse.

[7] In fact, NP is restricted during training to only questions whose associated tables have fewer than a certain threshold of rows and columns due to computational constraints.

answer selection; it is not immediately clear how to constrain predictions for one question based on the probability distribution over table cells from the previous question in the sequence.

To more fairly compare DynSP to the baseline systems, we also experiment with a "concatenated questions" setting, which allows the baselines to access sequential context. Here, we treat concatenated question prefixes of a sequence as additional training examples, where a question prefix includes all questions prior to the current question in the sequence.

For example, suppose the question sequence is: *1. what are all of the teams? 2. of those, which won championships?* For the second question, in addition to the original question–answer pair, we add the concatenated question sequence *"what are all of the teams? of those, which won championships?"* paired with the second question's answer. We refer to these concatenated question baselines as $\texttt{FP}^{+}$ and $\texttt{NP}^{+}$.

4.4.1 DynSP implementation details

Unlike previous dynamic neural network frameworks (Andreas et al., 2016; Looks et al., 2017), where each example can have different but *predetermined* structure, DynSP needs to dynamically explores and constructs different neural network structures for each question. Therefore, we choose DyNet (Neubig et al., 2017) as our implementation platform for its flexibility in composing computation graphs. We optimize our model parameters using standard stochastic gradient descent. The word embeddings are initialized with 100-d pretrained GloVe vectors (Pennington et al., 2014) and fine-tuned during training with dropout rate 0.5. For follow-up questions, we choose uniformly at random to use either

gold answers to the previous question or the model's previous predictions.[8] We constrain the maximum length of actions to 3 for computational efficiency and set the beam size to 15 in our reported models, as accuracy gains are negligible with larger beam sizes. We train our model for 30 epochs, although the best model on the validation set is usually found within the first 20 epochs. Only CPU is used in model training, and each epoch in the beam size 15 setting takes about 30 minutes to complete.

4.4.2 Results & Analysis

Table 4.4.2 shows the results of the baseline systems as well as our method on SQA's test set. For each system, we show both the overall accuracy, the sequence accuracy (the percentage of sequences for which every question was answered correctly), and the accuracy at each position in the sequence. Our method without any sequential information (`DynSP`) outperforms the standard baselines, and when the *subsequent* action is added (`DynSP`*), we improve both overall and sequence accuracy over the concatenated-question baselines.

With that said, all of the systems struggle to answer all questions within a sequence correctly, despite the fact that each individual question is simpler on average than those in WTQ. Most of the errors made by our system are due to either semantic matching challenges or limitations of the underlying parse language. In the middle example of Figure 4.4.2, the first question asks for a list of super heroes; from the model's point of view, *Real name* is a more relevant column than *Character*, although the latter is correct. The second question also contains a challenging matching problem where the unlisted

[8] Only predicted answers are used at test time.

Model	All	Seq	Pos 1	Pos 2	Pos 3
FP	34.1	7.2	52.6	25.6	**25.9**
NP	39.4	10.8	58.9	35.9	24.6
DynSP	42.0	10.2	**70.9**	35.8	20.1
FP$^+$	33.2	7.7	51.4	22.2	22.3
NP$^+$	40.2	11.8	60.0	35.9	25.5
DynSP*	**44.7**	**12.8**	70.4	**41.1**	23.6

Table 4.2: Accuracies of all systems on SQA; the models in the first half of the table treat questions independently, while those in the second half consider sequential context. Our method outperforms existing ones both in terms of overall accuracy as well as sequence accuracy.

home worlds referred to in the question are marked as *Unknown* in the table. Many of these matching issues are resolved by humans using common sense, which for computers requires far more data than is available in SQA to learn.

Even when there are no tricky discrepancies between question and table text, questions are often complex enough that their semantic parses cannot be expressed in our parse language. Although trivial on the surface, the final question in the bottom sequence of Figure 4.4.2 is one such example; the correct semantic parse requires access to the answers of both the first and second question, actions that we have not currently implemented in our language due to concerns with the search space size. Increasing the number of complex actions requires designing smarter optimization procedures, which we leave to future work.

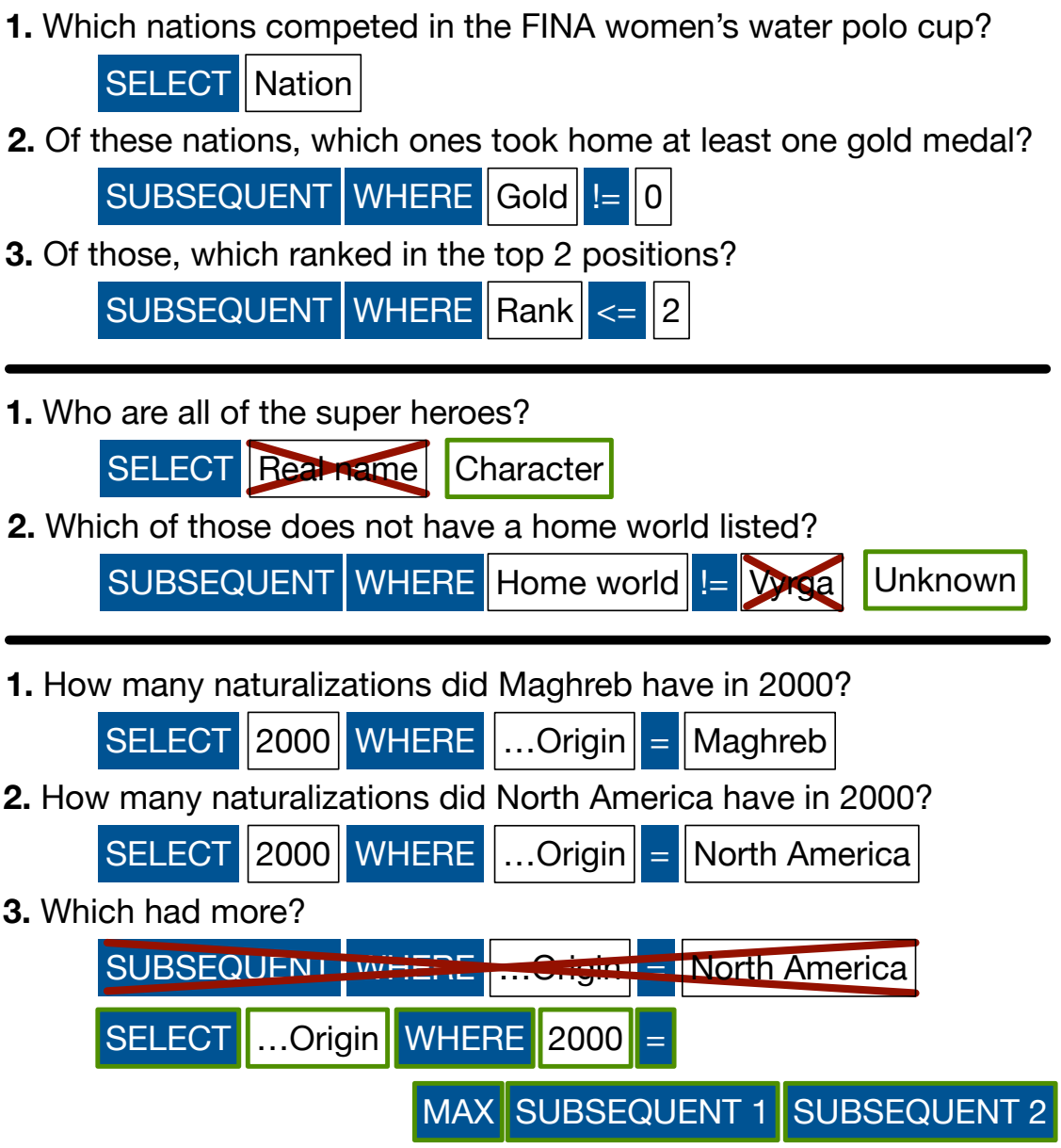


Figure 4.3: Parses computed by `DynSP` for three test sequences (actions in blue boxes, values from table in white boxes). *Top*: all three questions are parsed correctly. *Middle*: semantic matching errors cause the model to select incorrect columns and conditions. *Bottom*: The final question is unanswerable due to limitations of our parse language.

4.5 Related Work

Previous work on conversational QA has focused on small, single-domain datasets. Perhaps most related to our task is the context-dependent sentence analysis described in (Zettlemoyer and Collins, 2009), where conversations between customers and travel agents are mapped to logical forms after resolving referential expressions. (Artzi and Zettlemoyer, 2011) use another dataset of travel booking conversations to learn a semantic parser for complicated queries given user clarifications. More recently, Long et al. (2016) collect three contextual semantic parsing datasets (from synthetic domains) that contain coreferences to entities and actions. We differentiate ourselves from these prior works in two significant ways: first, our dataset is not restricted to a particular domain, and second, a major goal of our work is to analyze the different types of sequence progressions people create when they are trying to express a complicated intent.

Complex, interactive QA tasks have also been proposed in the information retrieval community, where the data source is a corpus of newswire text (Kelly and Lin, 2007). We also build on aspects of some existing interactive question-answering systems. For example, Harabagiu et al. (2005) include a module that predicts what a user will ask next given their current question.

Other than `FP` and `NP`, the work of Neural Symbolic Machines (NSM) (Liang et al., 2017) is perhaps the closest to ours. NSM aims to generate formal semantic parses of questions that can be executed on Freebase to retrieve answers, and is trained using the REINFORCE algorithm (Williams, 1992) augmented with approximate gold parses found in a separate curriculum learning stage. In comparison, finding reference parses is an

integral part of our algorithm. Our non-probabilistic, margin-based objective function also helps avoid the need for empirical tricks to handle normalization and proper sampling, which are crucial when applying REINFORCE in practice.

4.6 Conclusion & Future Work

In this work we move towards a conversational, multi-turn QA scenario in which systems must rely on prior context to answer the user's current question. To this end, we introduce SQA, a dataset that consists of 6,066 unique sequences of inter-related questions about Wikipedia tables, with 17,553 questions-answer pairs in total. To the best of our knowledge, SQA is the first semantic parsing dataset that addresses sequential question answering. We propose `DynSP`, a dynamic neural semantic parsing framework, for solving SQA. By formulating semantic parsing as a state–action search problem, our method learns modular neural network models through reward-guided search. `DynSP` outperforms existing state-of-the-art systems designed for answering complex questions when applied to SQA, and increases the gain after incorporating the subsequent actions.

In the future, we plan to investigate several interesting research questions triggered by this work. For instance, although our current formal language design covers most question types in SQA, it is nevertheless important to extend it further to make the semantic parser more robust (e.g., by including UNION or allowing comparison of multiple previous answers). Practically, allowing a more complicated semantic parse structure—either by increasing the number of primitive statements or the length of the parse—poses serious computational challenges in both model learning and inference. Because of the dynamic

nature of our framework, it is not trivial to leverage the computational capabilities of GPUs using minibatched training; we plan to investigate ways to take full advantage of modern computing machinery in the near future. Finally, better resolution of semantic matching errors is a top priority, and unsupervised learning from large external corpora is one way to make progress in this direction.

Chapter 5

Dynamically Modeling Fictional Relationships

The quiz bowl questions in Chapter 3 contain paragraph-length contexts, while the conversational histories in the sequential QA problem of Chapter 4 average about three sentences in length. Many language domains contain contexts that are substantially longer and more complicated. Consider novels, which contain character-centric narratives; relationships between characters develop chapter by chapter, and events in the story often have huge impact on these relationships. In this chapter, we consider the following question: how do we build neural networks that can produce valuable insights from just the raw texts of novels without any annotated data?[1]

5.1 Motivation

When two characters in a book break bread, is their meal just a result of biological needs or does it mean more? Cognard-Black et al. (2014) argue that this simple interaction reflects the diversity and background of the characters, while Foster (2009) suggests that the tone of a meal can portend either good or ill for the rest of the book. To support such theories, scholars use their literary expertise to draw connections between disparate books: Gabriel Conroy's dissonance from his family at a sumptuous feast in Joyce's *The Dead*, the frustration of Tyler's mother in *Dinner at the Homesick Restaurant*, and the grudging

[1] This chapter covers models and datasets previously proposed in Iyyer et al. (2016, NAACL).

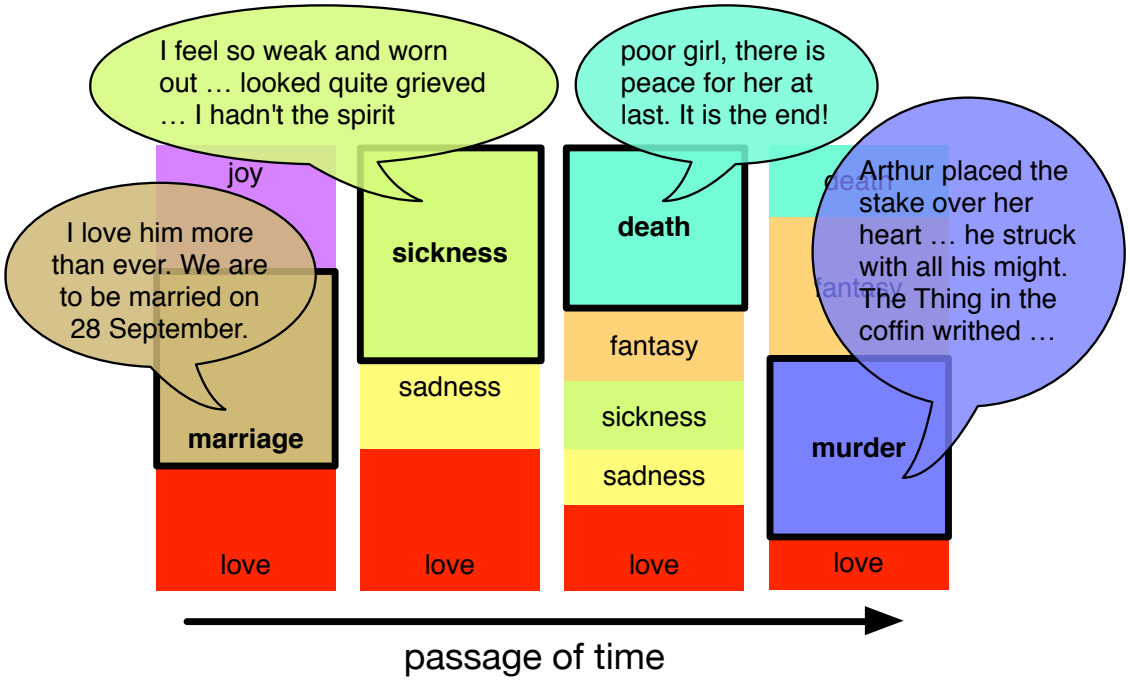


Figure 5.1: An example trajectory depicting the dynamic relationship between Lucy and Arthur in Bram Stoker's *Dracula*, which starts with love and ends with Arthur killing the vampiric Lucy. Each column describes the relationship state at a particular time by weights over a set of descriptors (larger weights shown as bigger boxes). Our goal is to learn—without supervision—both the descriptors and the trajectories from raw fictional texts.

respect for a blind man eating meatloaf in Carver's *Cathedral*.

However, these insights do not come cheap. It takes years of careful reading and internalization to make connections across books, which means that relationship symmetries and archetypes are likely to remain hidden in the millions of books published every year unless literary scholars are actively searching for them.

Natural language processing techniques have been increasingly used to assist in these literary investigations by discovering patterns in texts (Jockers, 2013). In Section 5.6 we review existing techniques that classify or cluster relationships between characters in books using a fixed set of labels (e.g., friend or enemy). However, such approaches

ignore interactions between characters that lie outside of the established lexicon and cannot account for the dynamic nature of relationships that evolve through the course of a book, such as the vampiric downfall of Lucy and Arthur's engagement in *Dracula* (Figure 5) or Winston Smith's rat-induced betrayal of Julia in *1984*.

To address these issues, we propose the task of unsupervised relationship modeling, in which a model jointly learns a set of *relationship descriptors* as well as *relationship trajectories* for pairs of literary characters. Instead of assigning a single descriptor to a particular relationship, the trajectories learned by the model are sequences of descriptors as in Figure 5.

The Bayesian *hidden topic Markov model* (**HTMM**) of Gruber et al. (2007) emerges as a natural choice for our task because it is capable of computing relationship descriptors (in the form of topics) and has an additional temporal component. However, our experiments show that the descriptors learned by the **HTMM** are not coherent and focus more on events or environments (e.g., meals, outdoors) than interpersonal states like happiness and sadness.

Motivated by recent advances in deep learning, we propose the *relationship modeling network* (**RMN**), which is a novel variant of a deep recurrent autoencoder that incorporates dictionary learning to learn relationship descriptors. We show that the **RMN** achieves better descriptor coherence and trajectory accuracy than the **HTMM** and other topic model baselines in two crowdsourced evaluations described in Section 5.4. In Section 5.5 we show qualitative results and make connections to existing literary scholarship.

5.2 A Dataset of Character Interactions

Our dataset consists of 1,383 fictional works pulled from Project Gutenberg and other Internet sources. Project Gutenberg has a limited selection (outside of science fiction) of mostly classic literature, so we add more contemporary novels from various genres such as mystery, romance, and fantasy to our dataset.

To identify character mentions, we run the Book-NLP pipeline of Bamman et al. (2014), which includes character name clustering, quoted speaker identification, and coreference resolution.[2] For every detected character mention, we define a span as beginning 100 tokens before the mention and ending 100 tokens after the mention. We do not use sentence or paragraph boundaries because they vary considerably depending on the author (e.g., William Faulkner routinely wrote single sentences longer than many of Hemingway's paragraphs). All spans in our dataset contain mentions to exactly two characters. This is a rather strict requirement that forces a reduction in data size, but spans in which more than two characters are mentioned are generally noisier.

Once we have identified usable spans in the dataset, we apply a second filtering step that removes relationships containing fewer than five spans. Without this filter, our dataset is dominated by fleeting interactions between minor characters; this is undesirable since our focus is on longer, mutable relationships. Finally, we filter our vocabulary by removing the 500 most frequently occurring words, as well as all words that occur in fewer than 100 books. The latter step helps correct for variation in time period and genre (e.g., "thou" and

[2]While this pipeline works reasonably well, it is unreliable for first-person narratives; we leave the necessary improvements to character name clustering, which are further expanded upon in Vala et al. (2015), for future work.

"thy" found in older works like the *Canterbury Tales*). Our final dataset contains 20,013 relationships and 380,408 spans, while our vocabulary contains 16,223 words.[3]

5.3 Relationship Modeling Networks

This section mathematically describes how we apply the **RMN** to relationship modeling on our dataset. Our model is similar in spirit to topic models: for an input dataset, the output of the **RMN** is a set of relationship descriptors (topics) and—for each relationship in the dataset—a trajectory, or a sequence of probability distributions over these descriptors (document-topic assignments). However, the **RMN** uses recent advances in deep learning to achieve better control over descriptor coherence and trajectory smoothness (Section 5.4).

5.3.1 Formalizing the Problem

Assume we have two characters c_1 and c_2 in book b. We define S_{c_1,c_2} as a sequence of token spans where each span $s_t \in S_{c_1,c_2}$ is itself a set of tokens $\{w_1, w_2, \ldots, w_l\}$ of fixed size l that contains mentions (either directly or by coreference) to both c_1 and c_2. In other words, S_{c_1,c_2} includes the text of every scene, chronologically ordered, in which c_1 and c_2 are present together.

5.3.2 Model Description

As in other neural network models for natural language processing, we begin by associating each word type w in our vocabulary with a real-valued embedding $\boldsymbol{v}_w \in \mathbb{R}^d$. These

[3]Code and span data available at http://github.com/miyyer/rmn.

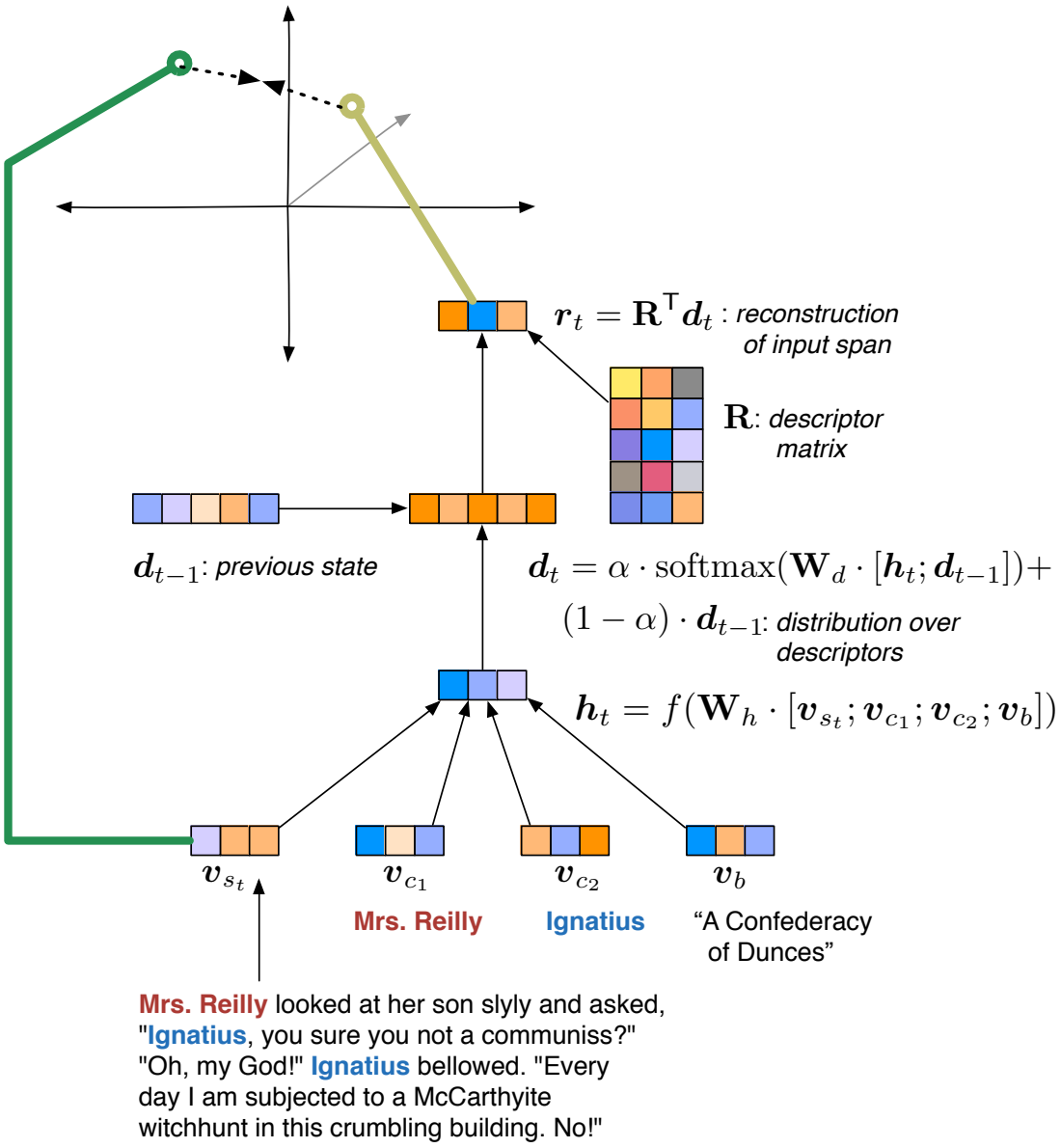


Figure 5.2: An example of the RMN's computations at a single time step. The model approximates the vector average of an input span (v_{s_t}) as a linear combination of descriptors from $\mathbf{R}$. The descriptor weights d_t define the relationship state at each time step and—when viewed as a sequence—form a relationship trajectory.

embeddings are rows of a $V \times d$ matrix $\mathbf{L}$, where V is the vocabulary size. Similarly, characters and books have their own embeddings in rows of matrices $\mathbf{C}$ and $\mathbf{B}$. We want $\mathbf{B}$ to capture global context information (e.g., "Moby Dick" takes place at sea) and $\mathbf{C}$ to

capture immutable aspects of characters not related to their relationships (e.g., Javert is a police officer). Finally, the **RMN** learns embeddings for relationship descriptors, which requires a second matrix $\mathbf{R}$ of size $K \times d$ where K is the number of descriptors, analogous to the number of topics in topic models.

Each input to the **RMN** is a tuple that contains identifiers for a book and two characters, as well as the spans corresponding to their relationship: $(b, c_1, c_2, S_{c_1,c_2})$. Given one such input, our objective is to reconstruct S_{c_1,c_2} using a linear combination of relationship descriptors from $\mathbf{R}$ as shown in Figure 5.3; we now describe this process formally.

5.3.2.1 Modeling Spans with Vector Averages

We use the **DAN** architecture detailed in Chapter 2 for span representation; below are the specific details. We compute a vector representation for each span s_t in S_{c_1,c_2} by averaging the embeddings of the words in that span,

$$\boldsymbol{v}_{s_t} = \frac{1}{l} \sum_{w \in s_t} \boldsymbol{v}_w. \tag{5.1}$$

Then, we concatenate $\boldsymbol{v}_{s_t}$ with the character embeddings $\boldsymbol{v}_{c_1}$ and $\boldsymbol{v}_{c_2}$ as well as the book embedding $\boldsymbol{v}_b$ and feed the resulting vector into a standard feed-forward layer to obtain a hidden state $\boldsymbol{h}_t$,

$$\boldsymbol{h}_t = f(\mathbf{W}_h \cdot [\boldsymbol{v}_{s_t}; \boldsymbol{v}_{c_1}; \boldsymbol{v}_{c_2}; \boldsymbol{v}_b]). \tag{5.2}$$

In all experiments, the transformation matrix W_h is $d \times 4d$, and we set f to the ReLu function, $\text{ReLu}(x) = \max(0, x)$.

5.3.2.2 Approximating Spans with Relationship Descriptors

Now that we can obtain representations of spans, we move on to learning descriptors using a variant of dictionary learning (Olshausen and Field, 1997; Elad and Aharon, 2006), where our descriptor matrix $\mathbf{R}$ is the dictionary and we are trying to approximate input spans as a linear combination of items from this dictionary.

Suppose we compute a hidden state for every span s_t in S_{c_1,c_2} (Equation 5.2). Now, given an $\boldsymbol{h}_t$, we compute a weight vector $\boldsymbol{d}_t$ over K relationship descriptors with some composition function g, which is fully specified in the next section. Conceptually, each $\boldsymbol{d}_t$ is a *relationship state*, and a *relationship trajectory* is a sequence of chronologically-ordered relationship states as shown in Figure 5. After computing $\boldsymbol{d}_t$, we use it to compute a reconstruction vector $\boldsymbol{r}_t$ by taking a weighted average over relationship descriptors,

$$\boldsymbol{r}_t = \mathbf{R}^\mathsf{T} \boldsymbol{d}_t. \tag{5.3}$$

Our goal is to make $\boldsymbol{r}_t$ similar to $\boldsymbol{v}_{s_t}$. We use a contrastive max-margin objective function similar to previous work (Weston et al., 2011; Socher et al., 2014). We randomly sample spans from our dataset and compute the vector average $\boldsymbol{v}_{s_n}$ for each sampled span as in Equation 5.1. This subset of span vectors is N. The unregularized objective J is a hinge loss that minimizes the inner product between $\boldsymbol{r}_t$ and the negative samples while simultaneously maximizing the inner product between $\boldsymbol{r}_t$ and $\boldsymbol{v}_{s_t}$,

$$J(\theta) = \sum_{t=0}^{|S_{c_1,c_2}|} \sum_{n \in N} \max(0, 1 - \boldsymbol{r}_t \boldsymbol{v}_{s_t} + \boldsymbol{r}_t \boldsymbol{v}_{s_n}), \tag{5.4}$$

where θ represents the model parameters.

5.3.2.3 Computing Weights over Descriptors

What function should we choose for our composition function g to represent a relationship state at a given time step? On the face of it, this seems trivial; we can project h_t to K dimensions and then apply a softmax or some other nonlinearity that yields non-negative weights.[4] However, this method ignores the relationship states at previous time steps. To model the temporal aspect of relationships, we can add a recurrent connection,

$$\boldsymbol{d}_t = \text{softmax}(\mathbf{W}_d \cdot [\boldsymbol{h}_t; \boldsymbol{d}_{t-1}]) \tag{5.5}$$

where $\mathbf{W}_d$ is of size $K \times (d + K)$ and $\text{softmax}(\boldsymbol{q}) = {}^{\exp \boldsymbol{q}}/_{\sum_{j=1}^{k} \exp \boldsymbol{q}_j}$.

Our hope is that this recurrent connection will carry some of the previous relationship state over to the current time step. It should be unlikely for two characters in love at time t to fall out of love at time $t + 1$ even if s_{t+1} does not include any love-related words. However, because the objective function in Equation 5.4 maximizes similarity with the current time step's input, the model is not forced to learn a smooth interpolation between the previous state and the current one. A natural remedy is to have the model predict the *next* time step's input instead, but this proves hard to optimize.

We instead *force* the model to use the previous relationship state by modifying

[4] We experiment with a variety of nonlinearities but find that the softmax yields the most interpretable results due to its predisposition to select a single descriptor.

Equation 5.5 to include a linear interpolation between $\boldsymbol{d}_t$ and $\boldsymbol{d}_{t-1}$,

$$\begin{aligned} \boldsymbol{d}_t = \alpha \cdot \text{softmax}(\mathbf{W}_d \cdot [\boldsymbol{h}_t; \boldsymbol{d}_{t-1}]) + \\ (1 - \alpha) \cdot \boldsymbol{d}_{t-1}. \end{aligned} \tag{5.6}$$

Here, α is a scalar between 0 and 1. We experiment with setting α to a fixed value of 0.5 as well as allowing the model to learn α as in

$$\alpha = \sigma(\boldsymbol{v}_\alpha^\mathsf{T} \cdot [\boldsymbol{h}_t; \boldsymbol{d}_{t-1}; \boldsymbol{v}_{s_t}]), \tag{5.7}$$

where σ is the sigmoid function and $\boldsymbol{v}_\alpha$ is a vector of dimensionality $2d + K$. Fixing $\alpha = 0.5$ initially and then tuning it after other parameters have converged improves training stability; for the specific hyperparameters we use see Section 5.4.[5]

5.3.2.4 Interpreting Descriptors and Enforcing Uniqueness

Recall that each descriptor is a d-dimensional row of $\mathbf{R}$. Because our objective function J forces these descriptors to be in the same vector space as that of the word embeddings $\mathbf{L}$, we can interpret them by looking at nearest neighbors in $\mathbf{L}$ using cosine distance as the similarity metric.

To discourage learning descriptors that are too similar to each other, we add another penalty term X to our objective function,

$$X(\theta) = \left\| \mathbf{R}\mathbf{R}^\mathsf{T} - \mathbf{I} \right\|, \tag{5.8}$$

[5]This strategy is reminiscent of alternative minimization strategies for dictionary learning (Agarwal et al., 2014), where the dictionary and weights are learned separately by keeping the other fixed.

where $\mathbf{I}$ is the identity matrix. This term comes from the component orthogonality constraint in independent component analysis (Hyvärinen and Oja, 2000).

We add J and X together to obtain our final training objective L,

$$L(\theta) = J(\theta) + \lambda X(\theta), \tag{5.9}$$

where λ is a hyperparameter that controls the magnitude of the uniqueness penalty.

5.4 Evaluating Descriptors and Trajectories

Because no previous work explores the interpretability of unsupervised relationship modeling over time, evaluating the **RMN** is tricky. Further compounding the problem is the subjective nature of the task; for example, is a trajectory that ignores a key event better than one that hallucinates episodes absent from source text?

With these issues in mind, we conduct three evaluations to show that our output is reasonable. First, we conduct a crowdsourced interpretability experiment that shows **RMN**s produce significantly more coherent descriptors than three topic model baselines. A second crowdsourced task indicates that our model produces trajectories that match plot summaries more accurately than topic models. Finally, we qualitatively compare the **RMN**'s output to existing static annotations of literary relationships and find both expected and surprising results.

5.4.1 Topic Model Baselines

Before moving onto the evaluations, we briefly describe three baseline models, all of which are Bayesian generative models. Latent Dirichlet allocation (Blei et al., 2003, **LDA**) learns a single document-topic distribution per document; we can apply **LDA** to our dataset by concatenating all spans from a relationship into a single document. Similarly, **NUBBI** (Chang et al., 2009a) learns separate sets of topics for relationships and individual characters.[6]

LDA and **NUBBI** are incapable of taking into account the chronological ordering of the spans because they view all relationships tokens as exchangeable. While we can compare the *descriptors* learned by these models to those of the **RMN**, we cannot evaluate their *trajectories*. We turn instead to the hidden topic Markov model (Gruber et al., 2007, **HTMM**), which foregoes the bag-of-words assumption of **LDA** and **NUBBI** in favor of modeling topic segments within a document as a Markov chain. This model outputs a smooth sequence of topic assignments over a document, so we can compare the *trajectories* it learns on our dataset to those of the **RMN**.

5.4.2 Experimental Settings

In our descriptor interpretability experiments, we vary the number of descriptors (topics) for all models ($K = 10, 30, 50$). We train **LDA** and **NUBBI** for 100 iterations with a collapsed Gibbs sampler, and the **HTMM** uses the default setting of 100 EM iterations.

For the **RMN**, we initialize the word embedding matrix $\mathbf{L}$ with 300-dimensional

[6] **NUBBI** requires additional spans that mention only a single character to differentiate character topics from relationship topics. None of the other models receives these extra data.

GloVe embeddings trained on the Common Crawl (Pennington et al., 2014). The character and book embeddings (**C** and **B**) are initialized randomly. We fix α to 0.5 for the first 15 epochs of training; after the descriptor matrix **R** has converged, we fix **R** and tune α using Equation 5.6 for 15 more epochs.[7] Since the topic model baselines do not have access to character and book metadata, for fair comparison we also train a "generic" version of the **RMN** (**GRMN**) where the metadata embeddings are removed from Equation 5.2. The uniqueness penalty λ is set to 10^{-4}.

All of the **RMN** model parameters except **L** are optimized using Adam (Kingma and Ba, 2014) with a learning rate of 0.001 for 30 epochs; the word embeddings are not fine-tuned during training.[8] We also apply word dropout (see Section 3.2.1.1) to the input spans, removing words from the vector average computation in Equation 5.1 with probability 0.5.

5.4.3 Do Descriptors Make Sense?

The goal of our first experiment is to compare the descriptors **R** learned by the **RMN** to the topics learned by the topic model baselines. We conduct a word intrusion experiment (Chang et al., 2009b): workers identify an "intruder" word from a set of words that—other than the intruder—come from the same topic. For the topic models, the five most probable words are joined by a highly-probable word from a different topic as the intruder. We use the same procedure for the **RMN** and **GRMN**, except that cosine similarity to descriptor embeddings replaces topic-word probability. To control for randomness in

[7] Preliminary experiments show that learning α and **R** simultaneously results in less interpretable descriptors.

[8] Tuning **L** ruins descriptor interpretability; pretrained embeddings are likely already a good solution for our problem.

RMN			HTMM		
Label	**MP**	**Nearest Neighbors**	**Label**	**MP**	**Most Probable Words**
sadness	1.0	regretful rueful pity pained despondent	violence	1.0	sword shot blood shouted swung
love	1.0	love delightful happiness enjoyed	boats	1.0	ship boat captain deck crew
murder	1.0	autopsy arrested homicide murdered	food	1.0	kitchen mouth glass food bread
worship	0.1	toil pray devote yourselves gather	sci-fi	0.0	suppose earth robots computer certain
moodiness	0.3	glumly snickered quizzically guiltily	fantasy	0.0	agreed magician dragon castle talent
informal	0.4	kinda damn heck guess shitty	military	0.1	ship captain lucky hour general

Table 5.1: Three high-precision (top) and three low-precision (bottom) descriptors for the **RMN** and **HTMM**, along with labels from an external evaluator and model precision (MP) computed via word intrusion experiments. The **RMN** is able to learn a variety of interpersonal states (e.g., love, sadness), while the **HTMM**'s most coherent topics are about concrete objects or events.

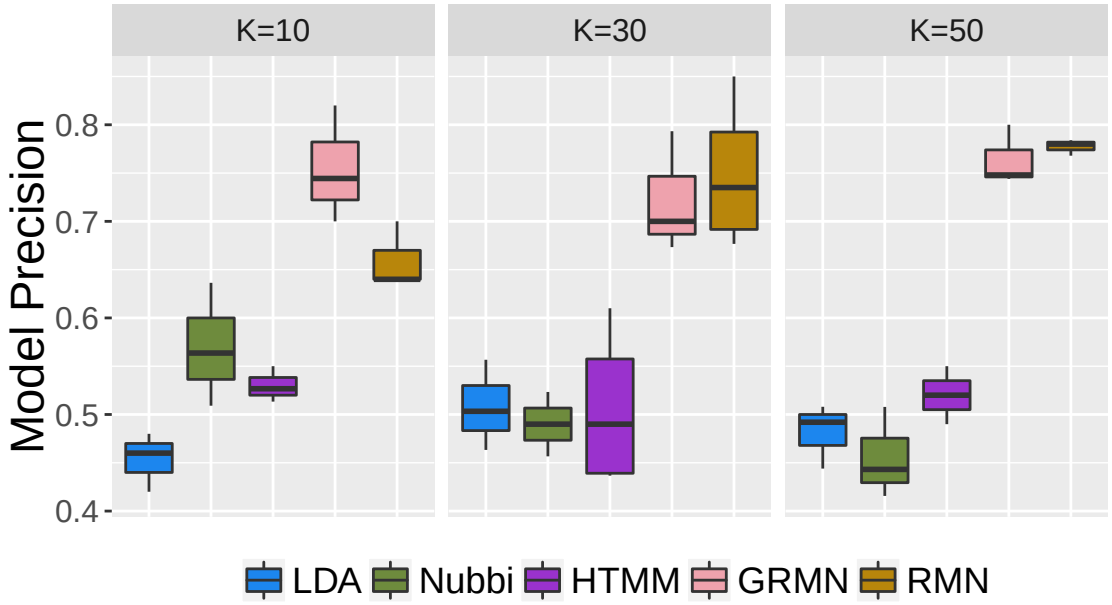


Figure 5.3: Model precision results from our word intrusion task. The **RMN** learns more interpretable descriptors than three topic model baselines.

the training process, we train three of each model, so the final experiment consists of 1,350 tasks ($K = 10, 30, 50$ descriptors per trial, three trials per model).

We collect judgments from ten different workers for each task using the Crowdflower platform.[9] Our evaluation metric, model precision (MP), is the fraction of workers that select the correct intruder word for a descriptor k. Low model precision signals descriptors that lack cohesive themes.

On average, the **RMN**'s descriptors are much more interpretable than those of the baselines, as it achieves a mean model precision of 0.73 (Figure 5.4.3) across all values of K. There is little difference between the model precision of the three topic model baselines, which hover around 0.5. There is also little difference between the **GRMN** and **RMN**; however, visualizing the learned character and book embeddings as in Figure 5.5 may be insightful for literary scholars. We show example high and low precision descriptors for the **RMN** and **HTMM** in Table 5.4.2; a full list is included in the supplementary material.

5.4.4 Do Trajectories Make Sense?

While the previous evaluation focused only on descriptor quality, our next experiment compares the trajectories learned by the best **RMN** model from the intrusion experiment (measured by highest mean model precision) to those learned by the best **HTMM** model, which is the only baseline capable of learning relationship trajectories. Workers read a plot summary and choose which model's trajectory best represents the relationship in question. We use the $K = 30$ setting because it provides the best balance between descriptor variety and trajectory interpretability.

[9] http://www.crowdflower.com

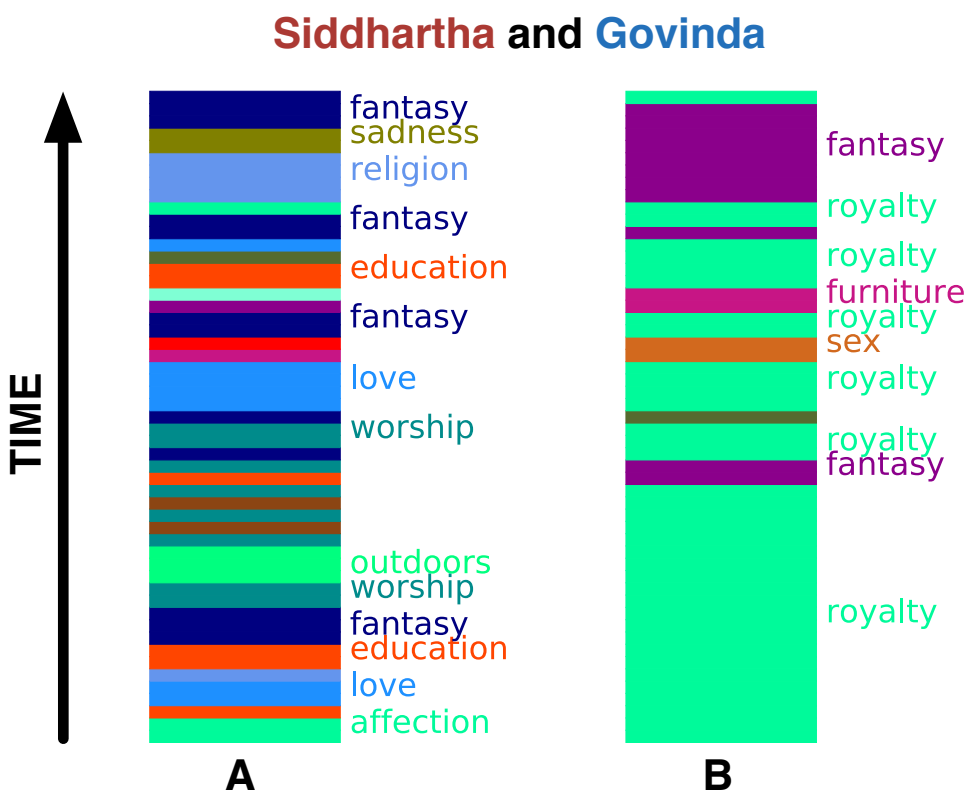
For this evaluation, we crawl Wikipedia, Goodreads, and SparkNotes for plot summaries associated with our 1,383 books. We then remove all relationships where each involved character is not mentioned at least five times in the summary, which results in a final evaluation set of 125 relationships.[10] We present workers with two characters, a plot summary, and a visualization of trajectories learned by the **RMN** and the **HTMM** (Figure 5.4.4). The workers then select the trajectory that best matches the relationship described by the summary.

To generate the visualizations, we first have an external annotator label each descriptor from both models with a single word as in Table 5.4.2. For fairness, the annotator is unaware of the underlying models. For the **RMN**, we visualize trajectories by displaying the label of the argmax over descriptor weights $\boldsymbol{d}_t$ at each time step t. Similarly, for the **HTMM**, we display the most probable topic at each time step.[11]

The results of this task with seven workers per comparison favor the **RMN**: for 87 out of the 125 evaluated relationships (69.6%), the workers choose the **RMN**'s trajectory over the **HTMM**'s. We compute the Fleiss κ value (Fleiss, 1971) to correct our inter-annotator agreement for chance and find that $\kappa = 0.32$, indicating fair agreement among the workers. Furthermore, thirty-four relationships had unanimous agreement among the seven workers; of these, twenty-six were unanimous in favor of the **RMN** compared to only eight for the **HTMM**.

[10] Without this filtering step, workers do not have enough information to compare the two models since most of the characters in our dataset are not mentioned in summaries.

[11] To reduce visual clutter, we ignore descriptors that persist for only a single time step.



Summary: **Govinda** is **Siddhartha**'s best friend and sometimes his follower. Like **Siddhartha**, **Govinda** devotes his life to the quest for understanding and enlightenment. He leaves his village with **Siddhartha** to join the Samanas, then leaves the Samanas to follow Gotama. He searches for enlightenment independently of **Siddhartha** but persists in looking for teachers who can show him the way. In the end, he is able to achieve enlightenment only because of **Siddhartha**'s love for him.

Figure 5.4: An example from the Crowdflower summary matching task; workers are asked to choose the trajectory (here, "A" is generated by the **RMN** and "B" by the **HTMM**) that best matches a provided summary that describes the relationship between Siddartha and Govinda (from *Siddartha* by Hesse).

5.4.5 What Makes a Relationship Positive?

While the previous two experiments show that the **RMN** is more interpretable and accurate than baseline models, we have not yet shown that its insights can aid in drawing connections

across various books and genres. As a first step in this direction, we investigate what makes a relationship positive or negative by comparing trajectories from the **RMN** and **HTMM** to static affinity annotations from a recently-released dataset (Massey et al., 2015) of fictional relationships. Expected correlations (e.g., murder and sadness are strongly negative descriptors) emerge alongside surprising ones (politics is negative, religion is positive).

The affinity labeling task of Massey et al. (2015) requires workers to describe a given relationship as positive, negative, or neutral. We consider only non-neutral relationships for which two annotators agree on the affinity label and remove all books not present in our own dataset. This filtering step results in 120 relationships, 78% of which are positive and the remaining 22% negative.

Since the annotations are static, we first aggregate our trajectories across all time steps. We compute K-dimensional "average positive" and "average negative" weight vectors $\boldsymbol{a}_p$ and $\boldsymbol{a}_n$ by averaging the relationship states $\boldsymbol{d}_t$ for the **RMN** and the document-topic distributions for the **HTMM** across all time steps for relationships labeled with a particular affinity. Then, we compute the vector difference $\boldsymbol{a}_p - \boldsymbol{a}_n$ and sort it to produce a ranked list of descriptors, where descriptors with positive differences occur more frequently in positive relationships. Table 5.4.5 shows the most positive and most negative descriptors; of particular interest is the large negative weight associated with political relationships from both models.

Model	Positive	Negative
RMN	education, love, religion, sex	politics, murder, sadness, royalty
HTMM	love, parental, business, outdoors	love, politics, violence, crime

Table 5.2: Descriptors most characteristic of positive and negative relationships, computed using existing annotations. Compared to the **RMN**, the **HTMM** struggles to coherently characterize negative relationships. Interestingly, both models show negative predispositions for political relationships, perhaps due to genre bias or class differences.

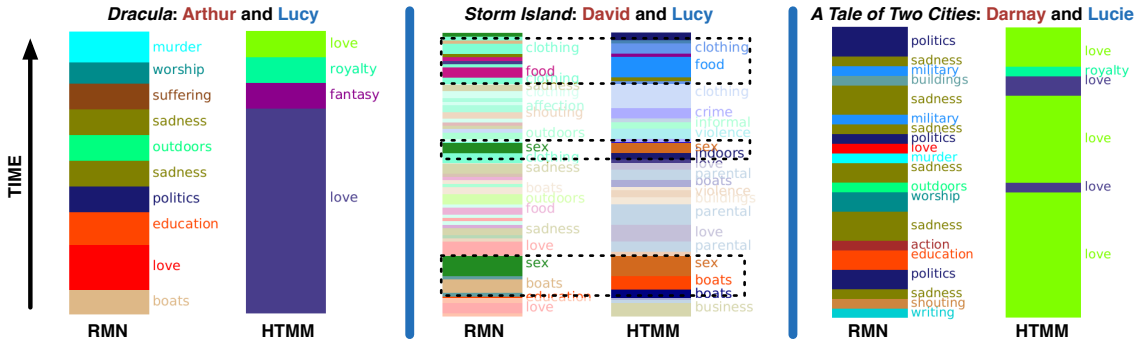


Figure 5.5: *Left*: the **RMN** is able to model Arthur and Lucy's trajectory reasonably well compared to our manually-created version in Figure 5. *Middle*: both models agree on event-based descriptors such as food and sex. *Right*: a failure case for the **RMN** in which it is unable to learn that Lucie Manette and Charles Darnay are in love.

5.5 Qualitative Analysis

Our experiments show the superiority of the **RMN** over various topic model baselines in both descriptor interpretability and trajectory accuracy, but what causes the improved performance? In this section, we analyze similarities between the **RMN** and **HTMM** and look at qualitative examples where the **RMN** succeeds and fails. We also connect the findings of our affinity experiment to existing literary scholarship.

Both models are equally proficient at learning and assigning event-based descriptors

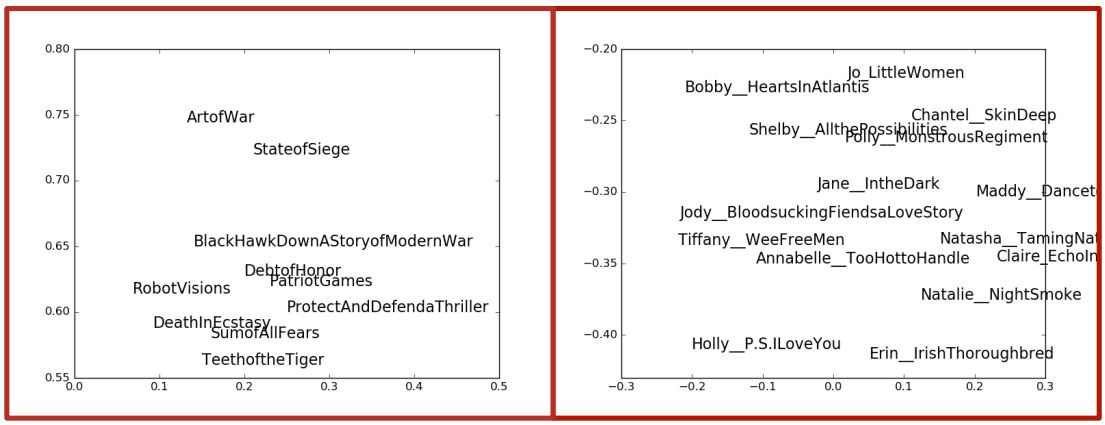


Figure 5.6: Clusters from PCA visualizations of the **RMN**'s learned book (left) and character (right) embeddings. We see a cluster of books about war and violence (many of which are authored by Tom Clancy) as well as a cluster of lead female characters from primarily romance novels. These visualizations show that the **RMN** can recover useful static representations of characters and books in addition to the dynamic relationship trajectories.

(e.g., crime, violence, food). More specifically, the **RMN** and **HTMM** agree on environmental descriptions (e.g., boats, outdoors) and graphic sexual scenes (Figure 5.5, middle).

However, the **RMN** is more sophisticated with interpersonal relationships. None of the topic model baselines learns negative emotional descriptors such as sadness or suffering, which explains the inaccurate **HTMM** trajectory of Arthur and Lucy in the left-most panel of Figure 5.5. All of the topic model baselines learn duplicate topics; in Table 5.4.5, one love descriptor is highly positive while a duplicate is strongly negative.[12] The **RMN** circumvents this problem with its uniqueness penalty (Equation 5.8).

While the increased descriptor variety is a positive, sometimes it leads the **RMN** astray. The model largely ignores the love between Charles Darnay and Lucie Manette in Dickens' *A Tale of Two Cities* due to book's sad tone; meanwhile, the **HTMM**'s trajectory,

[12]This "duplicate love" phenomenon persists even when we reduce the number of topics.

while vastly simplified, does pick up on the romance (Figure 5.5, right). While the **RMN**'s learnable book and character embeddings should help, the signal in a span cannot lead to the "proper" descriptor.

Both the **RMN** and **HTMM** learn that politics is strongly negative (Table 5.4.5). Existing scholarship supports this finding: Victorian-era authors, for example, are "obsessed with *otherness* . . . of antiquated social and legal institutions, and of autocratic and/or dictatorial abusive government" (Zarifopol-Johnston, 1995), while in science fiction, "dystopia——precisely because it is so much more common (than utopia)——bears the aspect of lived experience" (Gordin et al., 2010). Our affinity data comes primarily from Victorian novels (e.g., by Dickens and George Eliot), leading us to believe that that the models are behaving reasonably. Finally, returning to the "extra" meaning of meals discussed at the beginning of the chapter, food occurs *slightly* more frequently in positive relationships.

5.6 Related Work

There are two major areas upon which our work builds: computational literary analysis and deep neural networks for natural language processing.

Most previous work in computational literary analysis has focused either on characters or events. In the former category, graphical models and classifiers have been proposed for learning character personas from novels (Bamman et al., 2014; Flekova and Gurevych, 2015) and film summaries (Bamman et al., 2013). The **NUBBI** model of Chang et al. (2009a) learns topics that statically describe characters and their relationships. Because

these models lack temporal components (the focus of our task), we compare instead against the **HTMM** of Gruber et al. (2007).

Closest to our own work is the supervised structured prediction problem of Chaturvedi et al. (2016), in which features are designed to predict dynamic sequences of positive and negative interactions between two characters in plot summaries. Other research in this area includes social network construction from novels (Elson et al., 2010; Srivastava et al., 2016) and film (Krishnan and Eisenstein, 2015), as well as attempts to summarize and generate stories (Elsner, 2012).

While some of the relationship descriptors learned by our model are character-centric, others are more events-based, depicting actions rather than feelings; such descriptors have been the focus of much previous work (Schank and Abelson, 1977; Chambers and Jurafsky, 2008, 2009; Orr et al., 2014). Our model is more closely related to the plot units framework (Lehnert, 1981; Goyal et al., 2013), which annotates events with emotional states.

The **RMN** builds on deep recurrent autoencoders such as the hierarchical LSTM autoencoder of Li et al. (2015); however, it is more efficient because of the span-level vector averaging. It is also similar to recent neural topic model architectures (Cao et al., 2015; Das et al., 2015), although these models are limited to static document representations. We hope to apply the **RMN** to nonfictional datasets as well; in this vein, Iyyer et al. (2014b) apply a neural network to sentences from nonfiction political books for ideology prediction.

More generally, topic models and related generative models are a central tool for understanding large corpora from science (Talley et al., 2011) to politics (Nguyen et al., 2014). We show representation learning models like **RMN** can be just as interpretable as

LDA-based models. Other applications for which researchers have prioritized interpretable vector representations include text-to-vision mappings (Lazaridou et al., 2014) and word embeddings (Fyshe et al., 2015; Faruqui et al., 2015).

5.7 Conclusion

We formalize the task of unsupervised relationship modeling, which involves learning a set of relationship descriptors as well as a trajectory over these descriptors for each relationship in an input dataset. We present the **RMN**, a novel neural network architecture for this task that generates more interpretable descriptors and trajectories than topic model baselines. Finally, we show that the output of our model can lead to interesting insights when combined with annotations in an existing dataset.

Chapter 6

Understanding Panel-to-Panel Inferences in Comic Books

So far, we have looked at problems that involve understanding and extracting information from language-based contexts. To close out this thesis, I look at comic books, a multimodal domain that incorporates images and language into a single medium. As we will see, the network architectures are similar to those used in the previous chapters, and the challenges of leveraging information from previously-observed context still remain.[1]

Figure 6.1: Where did the snake in the last panel come from? Why is it biting the man? Is the man in the second panel the same as the man in the first panel? To answer these questions, readers form a larger meaning out of the narration boxes, speech bubbles, and artwork by applying closure across panels.

[1] This chapter includes content and figures from Iyyer et al. (2017a, CVPR).

6.1 Motivation

Comics are fragmented scenes forged into full-fledged stories by the imagination of their readers. A comics creator can condense anything from a centuries-long intergalactic war to an ordinary family dinner into a single panel. But it is what the creator *hides* from their pages that makes comics truly interesting: the unspoken conversations and unseen actions that lurk in the spaces (or gutters) between adjacent panels. For example, the dialogue in Figure 6 suggests that between the second and third panels, Gilda commands her snakes to chase after a frightened Michael in some sort of strange cult initiation. Through a process called *closure* (McCloud, 1994), which involves (1) understanding individual panels and (2) making connective inferences across panels, readers form coherent storylines from seemingly disparate panels such as these. In this chapter, we study whether computers can do the same by collecting a dataset of comic books (**COMICS**) and designing several tasks that require closure to solve.

Section 6.2 describes how we create **COMICS**,[2] which contains $\sim$1.2 million panels drawn from almost 4,000 publicly-available comic books published during the "Golden Age" of American comics (1938–1954). **COMICS** is challenging in both style and content compared to natural images (e.g., photographs), which are the focus of most existing datasets and methods (Krizhevsky et al., 2012; Xu et al., 2015; Xiong et al., 2016). Much like painters, comic artists can render a single object or concept in multiple artistic styles to evoke different emotional responses from the reader. For example, the lions in Figure 6.1 are drawn with varying degrees of realism: the more cartoonish lions, from humorous

[2]Data, code, and annotations available at http://github.com/miyyer/comics

comics, take on human expressions (e.g., surprise, nastiness), while those from adventure comics are more photorealistic.

Comics are not just visual: creators push their stories forward through text—speech balloons, thought clouds, and narrative boxes—which we identify and transcribe using optical character recognition (OCR). Together, text and image are often intricately woven together to tell a story that neither could tell on its own (Section 6.5). To understand a story, readers must connect dialogue and narration to characters and environments; furthermore, the text must be read in the proper order, as panels often depict long scenes rather than individual moments (Cohn, 2010). Text plays a much larger role in **COMICS** than it does for existing datasets of visual stories (Huang et al., 2016b).

To test machines' ability to perform closure, we present three novel cloze-style tasks in Section 6.6 that require a deep understanding of narrative and character to solve. In Section 6.7, we design four neural architectures to examine the impact of multimodality and contextual understanding via closure. All of these models perform significantly worse than humans on our tasks; we conclude with an error analysis (Section 6.8) that suggests future avenues for improvement.

6.2 Creating a dataset of comic books

Comics, defined by cartoonist Will Eisner as *sequential art* (Eisner, 1990), tell their stories in sequences of *panels*, or single frames that can contain both images and text. Existing comics datasets (Guérin et al., 2013; Matsui et al., 2015) are too small to train data-hungry machine learning models for narrative understanding; additionally, they lack diversity in

Figure 6.2: Different artistic renderings of lions taken from the **COMICS** dataset. The left-facing lions are more cartoonish (and humorous) than the ones facing right, which come from action and adventure comics that rely on realism to provide thrills.

visual style and genres. Thus, we build our own dataset, **COMICS**, by (1) downloading comics in the public domain, (2) segmenting each page into panels, (3) extracting textbox locations from panels, and (4) running OCR on textboxes and post-processing the output. Table 6.2.1 summarizes the contents of **COMICS**. The rest of this section describes each step of our data creation pipeline.

# Books	3,948
# Pages	198,657
# Panels	1,229,664
# Textboxes	2,498,657
Text cloze instances	89,412
Visual cloze instances	587,797
Char. coherence instances	72,313

Table 6.1: Statistics describing dataset size (top) and the number of total instances for each of our three tasks (bottom).

6.2.1 Where do our comics come from?

The "Golden Age of Comics" began during America's Great Depression and lasted through World War II, ending in the mid-1950s with the passage of strict censorship regulations. In contrast to the long, world-building story arcs popular in later eras, Golden Age comics tend to be small and self-contained; a single book usually contains multiple different stories sharing a common theme (e.g., crime or mystery). While the best-selling Golden Age comics tell of American superheroes triumphing over German and Japanese villains, a variety of other genres (such as romance, humor, and horror) also enjoyed popularity (Goulart, 2004). The Digital Comics Museum (DCM)[3] hosts user-uploaded scans of many comics by lesser-known Golden Age publishers that are now in the public domain due to copyright expiration. To avoid off-square images and missing pages, as the scans vary in resolution and quality, we download the 4,000 highest-rated comic books from DCM.[4]

[3] http://digitalcomicmuseum.com/

[4] Some of the panels in **COMICS** contain offensive caricatures and opinions reflective of that period in American history.

6.2.2 Breaking comics into their basic elements

The DCM comics are distributed as compressed archives of JPEG page scans. To analyze closure, which occurs from panel-to-panel, we first extract panels from the page images. Next, we extract textboxes from the panels, as both location and content of textboxes are important for character and narrative understanding.

Panel segmentation: Previous work on panel segmentation uses heuristics (Li et al., 2014) or algorithms such as density gradients and recursive cuts (Tanaka et al., 2007; Pang et al., 2014b; Rigaud et al., 2015) that rely on pages with uniformly white backgrounds and clean gutters. Unfortunately, scanned images of eighty-year old comics do not particularly adhere to these standards; furthermore, many DCM comics have non-standard panel layouts and/or textboxes that extend across gutters to multiple panels.

After our attempts to use existing panel segmentation software failed, we turned to deep learning. We annotate 500 randomly-selected pages from our dataset with rectangular bounding boxes for panels. Each bounding box encloses both the panel artwork and the textboxes within the panel; in cases where a textbox spans multiple panels, we necessarily also include portions of the neighboring panel. After annotation, we train a region-based convolutional neural network to automatically detect panels. In particular, we use Faster R-CNN (Ren et al., 2015) initialized with a pretrained `VGG_CNN_M_1024` model (Chatfield et al., 2014) and alternatingly optimize the region proposal network and the detection network. In Western comics, panels are usually read left-to-right, top-to-bottom, so we also have to properly order all of the panels within a page after extraction. We compute

the midpoint of each panel and sort them using Morton order (Morton, 1966), which gives incorrect orderings only for rare and complicated panel layouts.

Textbox segmentation: Since we are particularly interested in modeling the interplay between text and artwork, we need to also convert the text in each panel to a machine-readable format.[5] As with panel segmentation, existing comic textbox detection algorithms (Ho et al., 2012; Rigaud et al., 2013) could not accurately localize textboxes for our data. Thus, we resort again to Faster R-CNN: we annotate 1,500 panels for textboxes,[6] train a Faster-R-CNN, and sort the extracted textboxes within each panel using Morton order.

6.2.3 OCR

The final step of our data creation pipeline is applying OCR to the extracted textbox images. We unsuccessfully experimented with two trainable open-source OCR systems, Tesseract (Smith, 2007) and Ocular (Berg-Kirkpatrick et al., 2013), as well as Abbyy's consumer-grade FineReader.[7] The ineffectiveness of these systems is likely due to the considerable variation in comic fonts as well as domain mismatches with pretrained language models (comics text is always capitalized, and dialogue phenomena such as dialects may not be adequately represented in training data). Google's Cloud Vision OCR[8] performs much better on comics than any other system we tried. While it sometimes

[5] Alternatively, modules for text spotting and recognition (Jaderberg et al., 2016) could be built into architectures for our downstream tasks, but since comic dialogues can be quite lengthy, these modules would likely perform poorly.

[6] We make a distinction between *narration* and *dialogue*; the former usually occurs in strictly rectangular boxes at the top of each panel and contains text describing or introducing a new scene, while the latter is usually found in speech balloons or thought clouds.

[7] http://www.abbyy.com

[8] http://cloud.google.com/vision

struggles to detect short words or punctuation marks, the quality of the transcriptions is good considering the image domain and quality. We use the Cloud Vision API to run OCR on all 2.5 million textboxes for a cost of $3,000. We post-process the transcriptions by removing systematic spelling errors (e.g., failing to recognize the first letter of a word). Finally, each book in our dataset contains three or four full-page product advertisements; since they are irrelevant for our purposes, we train a classifier on the transcriptions to remove them.

6.3 OCR Post-Processing and Advertisement Removal

OCR makes systematic mistakes on our textboxes. We target two types of these mistakes using PyEnchant:[9] 1) where the OCR system fails to recognize the first letter of a particular word (e.g., *eleportation* instead of *teleportation*), and 2) where the OCR system transcribes part of a word as a single alphabetical character. To eliminate errors of the first type, we start by tokenizing the OCR output using NLTK's Punkt Tokenizer.[10] We then sort the vocabulary of the tokenized OCR output in decreasing order of frequency and pick words ranked from 10,001 to 100,000, because most misspelled words are also rare. For each of these words that is length three or longer, we look up the most likely suggestion offered by PyEnchant. If the only difference between the most likely suggestion and the original word is an additional letter in the first position of the suggestion, then we replace the word with the suggestion everywhere in our corpus. To correct the second type of errors, we simply delete all single character alphabetical tokens that are not one of *a, d, i, m, s, t*—characters

[9] http://pythonhosted.org/pyenchant/faq.html
[10] http://www.nltk.org/

that can plausibly occur by themselves quite frequently (some occur after an apostrophe).

In addition to spelling errors, the books in COMICS contain many advertisements that we need to remove before generating data for our tasks. While most dialogue and narration boxes contain less than 30 words, longer textboxes frequently come from full-page product advertisements (e.g., Figure 6.3). However, detecting ads from page images is not easy. Some ads are deceptively similar to comic pages, containing images and even containing faux mini-comics. Aside from ads, there are also other undesirable pages; many books contain text-only short stories in addition to comics. We remove these kinds of pages using features from OCR transcriptions. We annotate each page of 100 random books with a label indicating the presence or absence of an invalid page as our training set and each page of twenty random books as our test set. Out of 6,117 annotated pages, 697 of them are either advertisements or text-only stories (11.4%). We train a binary classifier using Vowpal Wabbit:[11] which takes the OCR text for all the panels of a pages as lexical features (unigrams and bigrams). We improve our model by adding features like total count of words in the page and a count of non-alphanumeric characters. Our model gives us a total misclassification error of 8% and a false negative error of 17.3%, which means it misses one invalid page out of every six. The model has a negligible false positive error of 0.2%. Using this model to filter the entire dataset of 198,657 pages yields 13,200 invalid pages.

6.4 Examples from Dataset Creation

OCR transcription is the final stage of our data creation pipeline (panel extraction → textbox extraction → OCR). Therefore, faulty outputs in any of the preceeding steps can

[11] https://github.com/JohnLangford/vowpal_wabbit/wiki

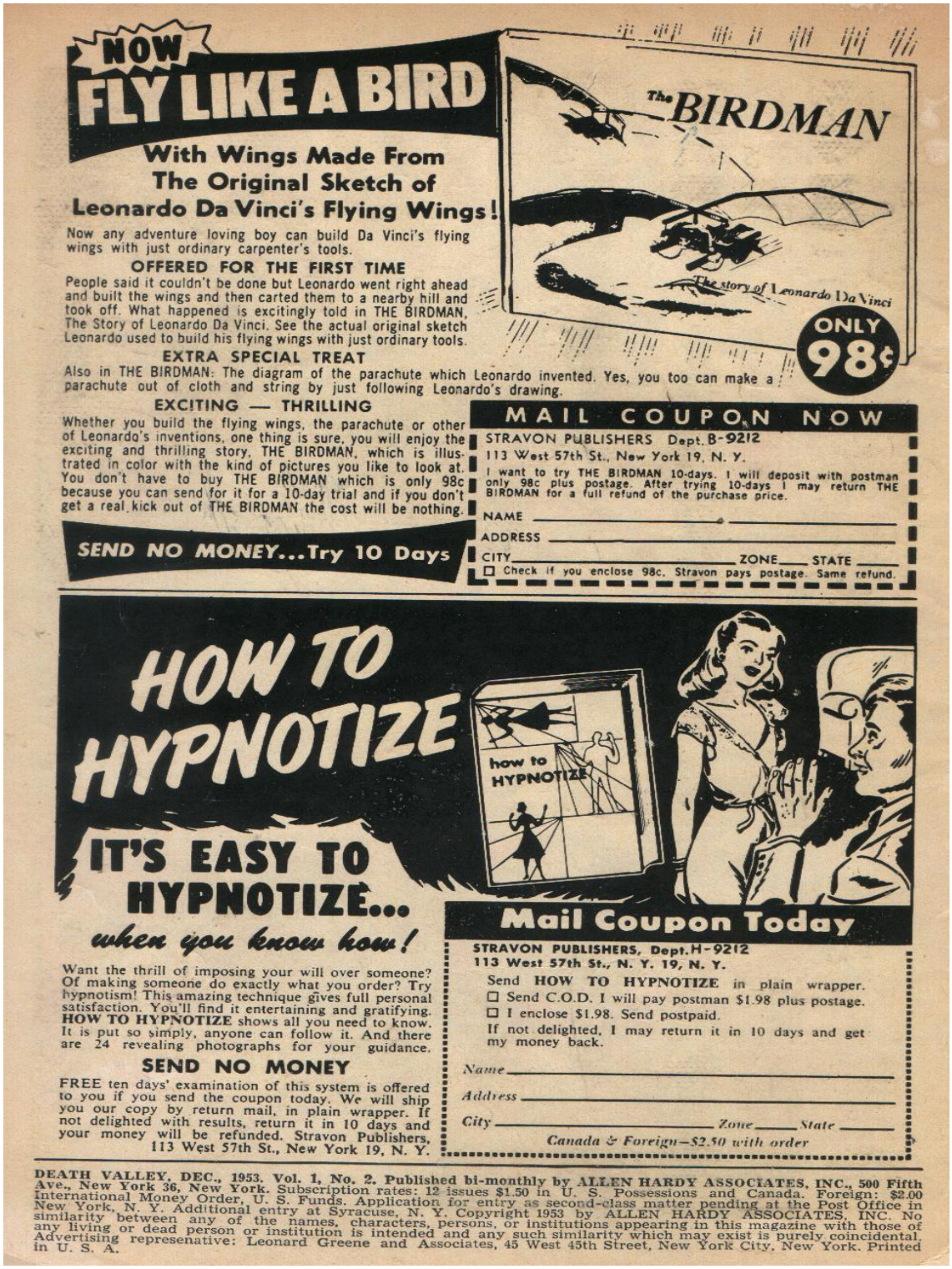


Figure 6.3: An advertisement from the dataset. The juxtaposition of text and image causes it to slightly resemble a comics page.

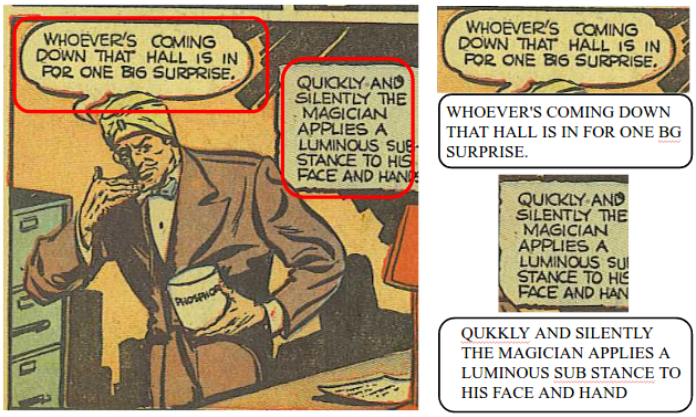
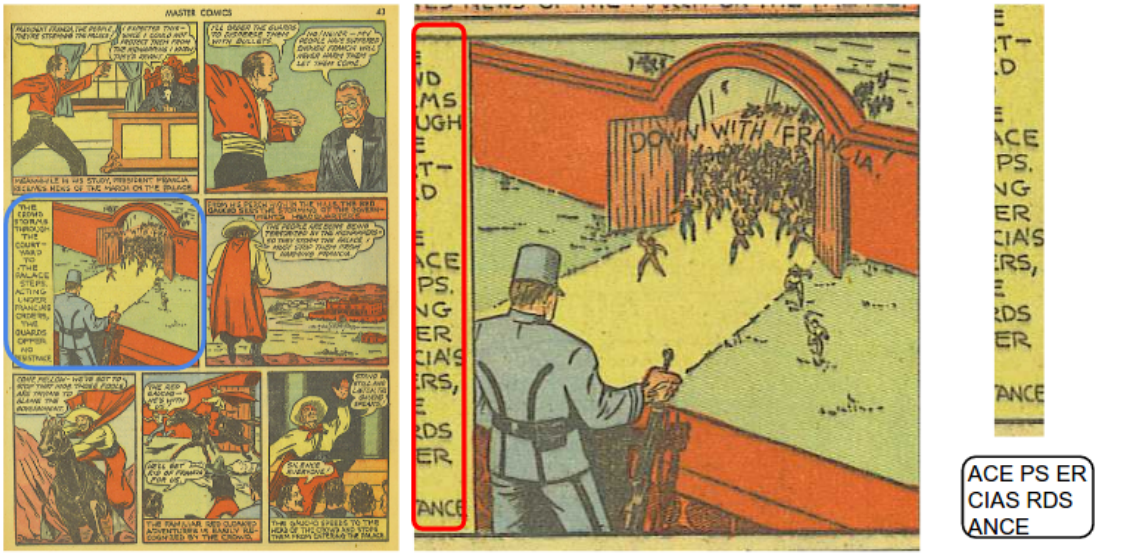


Figure 6.4: A minor OCR error. Mistakes such as predicting "BG" for "BIG" are understandable, since the 'I' in "BIG" is barely visible. Similarly, the "IC" in "QUICKLY" looks a lot like "K" in this font. Finally, "SUB STANCE" is predicted rather than "SUBSTANCE", due to an end-of-line word break.

lead to faulty OCR outputs. In Figure 6.4, there are only minor errors in OCR extraction due to understandable misinterpretations of the text in the dialog boxes. For example, the OCR interprets the letters "IC" as "K", which leads to incorrectly predicting the word "QUICKLY" as "QUKKLY". However, in Figure 6.5, we observe a more critical error due to missing pixels in the panel extraction process. Due to the layout of the textbox in the panel, crucial portions of the text are trimmed from view; while the OCR does a valiant job of predicting the contents of the textbox, its output is gibberish.

6.5 Data Analysis

In this section, we explore what makes understanding narratives in **COMICS** difficult, focusing specifically on *intrapanel* behavior (how images and text interact within a panel) and *interpanel* transitions (how the narrative advances from one panel to the next). We characterize panels and transitions using a modified version of the annotation scheme in



a. Page with ground truth panel location

b. Output of Panel Extraction

c. Textbox Extraction with OCR

Figure 6.5: A major OCR error. In part a) of the figure, note the location of the panel in the page. b) gives us the panel as predicted by the RCNN, but a critical portion of the text is missing. As a consequence, the textbox extraction is also faulty, rendering the OCR completely meaningless.

Scott McCloud's "Understanding Comics" (McCloud, 1994). Over 90% of panels rely on both text and image to convey information, as opposed to just using a single modality. Closure is also important: to understand most transitions between panels, readers must make complex inferences that often require common sense (e.g., connecting jumps in space and/or time, recognizing when new characters have been introduced to an existing scene). We conclude that any model trained to understand narrative flow in **COMICS** will have to effectively tie together multimodal inputs through closure.

To perform our analysis, we manually annotate 250 randomly-selected pairs of consecutive panels from **COMICS**. Each panel of a pair is annotated for intrapanel behavior, while an interpanel annotation is assigned to the transition between the panels. Two annotators independently categorize each pair, and a third annotator makes the final

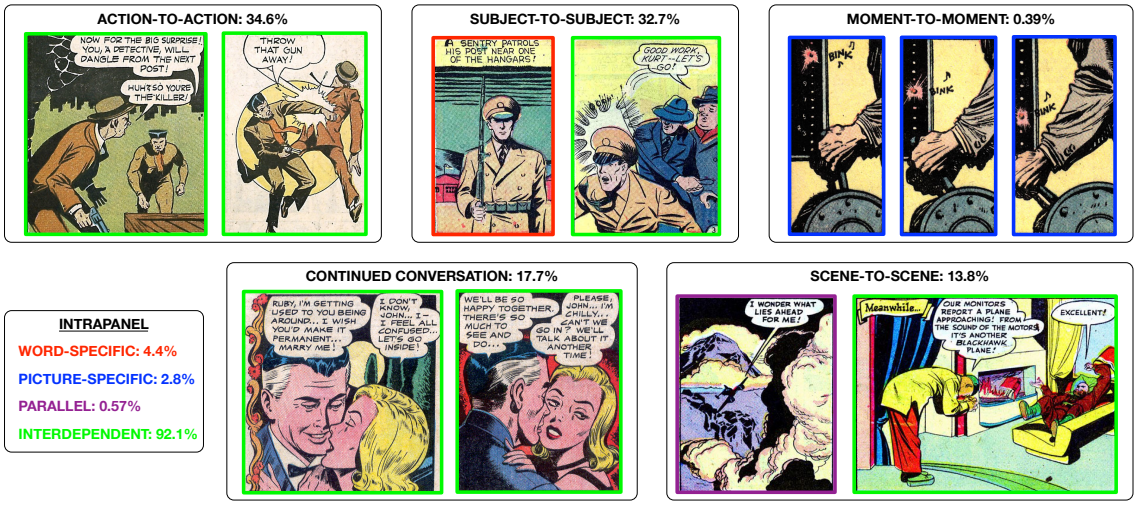


Figure 6.6: Five example panel sequences from **COMICS**, one for each type of interpanel transition. Individual panel borders are color-coded to match their intrapanel categories (legend in bottom-left). Moment-to-moment transitions unfold like frames in a movie, while scene-to-scene transitions are loosely strung together by narrative boxes. Percentages are the relative prevalance of the transition or panel type in an annotated subset of **COMICS**.

decision when they disagree. We use four intrapanel categories (definitions from McCloud, percentages from our annotations):

1. **Word-specific, 4.4%**: The pictures illustrate, but do not significantly add to a largely complete text.
2. **Picture-specific, 2.8%**: The words do little more than add a soundtrack to a visually-told sequence.
3. **Parallel, 0.6%**: Words and pictures seem to follow very different courses without intersecting.
4. **Interdependent, 92.1%**: Words and pictures go hand-in-hand to convey an idea that neither could convey alone.

We group interpanel transitions into five categories:

1. **Moment-to-moment, 0.4%**: Almost no time passes between panels, much like adjacent frames in a video.
2. **Action-to-action, 34.6%**: The same subjects progress through an action within the same scene.
3. **Subject-to-subject, 32.7%**: New subjects are introduced while staying within the same scene or idea.
4. **Scene-to-scene, 13.8%**: Significant changes in time or space between the two panels.
5. **Continued conversation, 17.7%**: Subjects continue a conversation across panels without any other changes.

The two annotators agree on 96% of the intrapanel annotations (Cohen's $\kappa = 0.657$), which is unsurprising because almost every panel is interdependent. The interpanel task is significantly harder: agreement is only 68% (Cohen's $\kappa = 0.605$). Panel transitions are more diverse, as all types except moment-to-moment are relatively common (Figure 6.5); interestingly, moment-to-moment transitions require the least amount of closure as there is almost no change in time or space between the panels. Multiple transition types may occur in the same panel, such as simultaneous changes in subjects and actions, which also contributes to the lower interpanel agreement.

6.6 Tasks that test closure

To explore closure in **COMICS**, we design three novel tasks (*text cloze*, *visual cloze*, and *character coherence*) that test a model's ability to understand narratives and characters

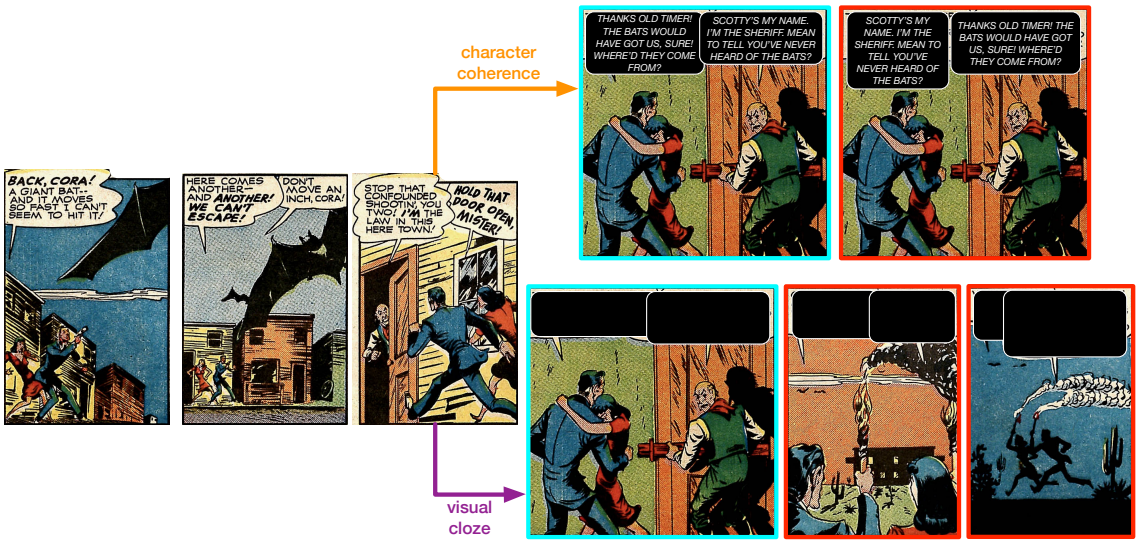


Figure 6.7: In the character coherence task (top), a model must order the dialogues in the final panel, while visual cloze (bottom) requires choosing the image of the panel that follows the given context. For visualization purposes, we show the original context panels; during model training and evaluation, textboxes are blacked out in every panel.

given a few panels of context. As shown in the previous section's analysis, a high percentage of panel transitions require non-trivial inferences from the reader; to successfully solve our proposed tasks, a model must be able to make the same kinds of connections.

While their objectives are different, all three tasks follow the same format: given preceding panels $p_{i-1}, p_{i-2}, \ldots, p_{i-n}$ as context, a model is asked to predict some aspect of panel p_i. While previous work on visual storytelling focuses on *generating* text given some context (Huang et al., 2016a), the dialogue-heavy text in **COMICS** makes evaluation difficult (e.g., dialects, grammatical variations, many rare words). We want our evaluations to focus specifically on closure, not generated text quality, so we instead use a cloze-style framework (Taylor, 1953): given c candidates—with a single correct option—models must use the context panels to rank the correct candidate higher than the others. The rest of this

section describes each of the three tasks in detail; Table 6.2.1 provides the total instances of each task with the number of context panels $n = 3$.

Text Cloze: In the *text cloze* task, we ask the model to predict what text out of a set of candidates belongs in a particular textbox, given both context panels (text and image) as well as the current panel image. While initially we did not put any constraints on the task design, we quickly noticed two major issues. First, since the panel images include textboxes, any model trained on this task could in principle learn to crudely imitate OCR by matching text candidates to the actual image of the text. To solve this problem, we "black out" the rectangle given by the bounding boxes for each textbox in a panel (see Figure 6.6).[12] Second, panels often have multiple textboxes (e.g., conversations between characters); to focus on interpanel transitions rather than intrapanel complexity, we restrict p_i to panels that contain only a single textbox. Thus, nothing from the current panel matters other than the artwork; the majority of the predictive information comes from previous panels.

Visual Cloze: We know from Section 6.5 that in most cases, text and image work interdependently to tell a story. In the *visual cloze* task, we follow the same set-up as in *text cloze*, but our candidates are images instead of text. A key difference is that models are not given text from the final panel; in *text cloze*, models are allowed to look at the final panel's artwork. This design is motivated by eyetracking studies in single-panel cartoons, which show that readers look at artwork before reading the text (Carroll et al., 1992), although

[12]To reduce the chance of models trivially correlating candidate length to textbox size, we remove very short and very long candidates.

atypical font style and text length can invert this order (Foulsham et al., 2016).

Character Coherence: While the previous two tasks focus mainly on narrative structure, our third task attempts to isolate character understanding through a re-ordering task. Given a jumbled set of text from the textboxes in panel p_i, a model must learn to match each candidate to its corresponding textbox. We restrict this task to panels that contain exactly two dialogue boxes (narration boxes are excluded to focus the task on characters). While it is often easy to order the text based on the language alone (e.g., "how's it going" always comes before "fine, how about you?"), many cases require inferring which character is likely to utter a particular bit of dialogue based on both their previous utterances and their appearance (e.g., Figure 6.6, top).

6.6.1 Task Difficulty

For *text cloze* and *visual cloze*, we have two difficulty settings that vary in how cloze candidates are chosen. In the *easy* setting, we sample textboxes (or panel images) from the entire **COMICS** dataset at random. Most incorrect candidates in the easy setting have no relation to the provided context, as they come from completely different books and genres. This setting is thus easier for models to "cheat" on by relying on stylistic indicators instead of contextual information. With that said, the task is still non-trivial; for example, many bits of short dialogue can be applicable in a variety of scenarios. In the *hard* case, the candidates come from nearby pages, so models must rely on the context to perform well. For *text cloze*, all candidates are likely to mention the same character names and entities, while color schemes and textures become much less distinguishing for *visual cloze*.

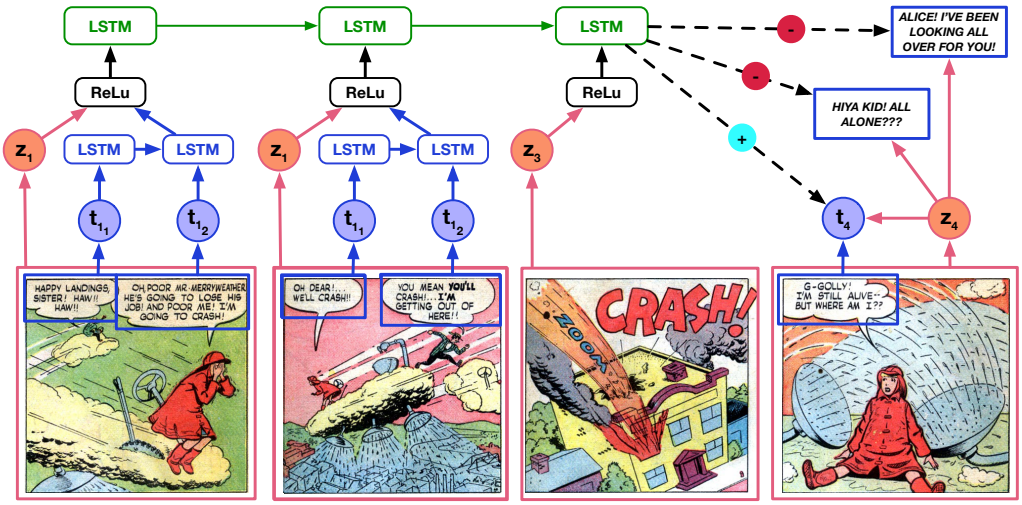


Figure 6.8: The **image-text** architecture applied to an instance of the *text cloze* task. Pretrained image features are combined with learned text features in a hierarchical LSTM architecture to form a context representation, which is then used to score text candidates.

6.7 Models & Experiments

To measure the difficulty of these tasks for deep learning models, we adapt strong baselines for multimodal language and vision understanding tasks to the comics domain. We evaluate four different neural models, variants of which were also used to benchmark the Visual Question Answering dataset (Antol et al., 2015) and encode context for visual storytelling (Huang et al., 2016b): *text-only*, *image-only*, and two *image-text* models. Our best-performing model encodes panels with a hierarchical LSTM architecture (see Figure 6.7).

On *text cloze*, accuracy increases when models are given images (in the form of pretrained VGG-16 features) in addition to text; on the other tasks, incorporating both modalities is less important. Additionally, for the *text cloze* and *visual cloze* tasks, models perform far worse on the *hard* setting than the *easy* setting, confirming our intuition that

these tasks are non-trivial when we control for stylistic dissimilarities between candidates. Finally, none of the architectures outperform human baselines, which demonstrates the difficulty of understanding **COMICS**: image features obtained from models trained on natural images cannot capture the vast variation in artistic styles, and textual models struggle with the richness and ambiguity of colloquial dialogue highly dependent on visual contexts. In the rest of this section, we first introduce a shared notation and then use it to specify all of our models.

6.7.1 Model definitions

In all of our tasks, we are asked to make a prediction about a particular panel given the preceding n panels as context.[13] Each panel consists of three distinct elements: image, text (OCR output), and textbox bounding box coordinates. For any panel p_i, the corresponding image is z_i. Since there can be multiple textboxes per panel, we refer to individual textbox contents and bounding boxes as t_{i_x} and b_{i_x}, respectively. Each of our tasks has a different set of answer candidates A: *text cloze* has three text candidates $t_{a_{1\ldots3}}$, *visual cloze* has three image candidates $z_{a_{1\ldots3}}$, and *character coherence* has two combinations of text / bounding box pairs, $\{t_{a_1}/b_{a_1}, t_{a_2}/b_{a_2}\}$ and $\{t_{a_1}/b_{a_2}, t_{a_2}/b_{a_1}\}$. Our architectures differ mainly in the encoding function g that converts a sequence of context panels $p_{i-1}, p_{i-2}, \ldots, p_{i-n}$ into a fixed-length vector c. We score the answer candidates by taking their inner product with c and normalizing with the softmax function,

$$s = \text{softmax}(A^T c), \tag{6.1}$$

[13] Test and validation instances for all tasks come from comic books that are unseen during training.

and we minimize the cross-entropy loss against the ground-truth labels.[14]

Text-only: The text-only baseline only has access to the text t_{i_x} within each panel. Our encoding function g processes this text on multiple levels: we first compute a representation for each t_{i_x} with a word embedding sum[15] and then combine multiple textboxes within the same panel using an *intrapanel* LSTM (Hochreiter and Schmidhuber, 1997). Finally, we feed the panel-level representations to an *interpanel* LSTM and take its final hidden state as the context representation (Figure 6.7). For *text cloze*, the answer candidates are also encoded with a word embedding sum; for *visual cloze*, we project the 4096-d `fc7` layer of VGG-16 down to the word embedding dimensionality with a fully-connected layer.[16]

Image-only: The image-only baseline is even simpler: we feed the `fc7` features of each context panel to an LSTM and use the same objective function as before to score candidates. For *visual cloze*, we project both the context and answer representations to 512-d with additional fully-connected layers before scoring. While the **COMICS** dataset is certainly large, we do not attempt learning visual features from scratch as our task-specific signals are far more complicated than simple image classification. We also try fine-tuning the lower-level layers of VGG-16 (Aytar et al., 2016); however, this substantially lowers task accuracy even with very small learning rates for the fine-tuned layers.

[14] Performance falters slightly on a development set with contrastive max-margin loss functions (Socher et al., 2014) in place of our softmax alternative.

[15] As in previous work for visual question answering (Zhou et al., 2015), we observe no noticeable improvement with more sophisticated encoding architectures.

[16] For training and testing, we use three panels of context and three candidates. We use a vocabulary size of 30,000 words, restrict the maximum number of textboxes per panel to three, and set the dimensionality of word embeddings and LSTM hidden states to 256. Models are optimized using Adam (Kingma and Ba, 2014) for ten epochs, after which we select the best-performing model on the dev set.

Model	Text Cloze		Visual Cloze		Char. Coheren.
	easy	*hard*	*easy*	*hard*	
Random	33.3	33.3	33.3	33.3	50.0
Text-only	63.4	52.9	55.9	48.4	68.2
Image-only	51.7	49.4	**85.7**	**63.2**	**70.9**
NC-image-text	63.1	59.6	-	-	65.2
Image-text	**68.6**	**61.0**	81.3	59.1	69.3
Human	–	84	–	88	87

Table 6.2: Combining image and text in neural architectures improves their ability to predict the next image or dialogue in **COMICS** narratives. The contextual information present in preceding panels is useful for all tasks: the model that only looks at a single panel (**NC-image-text**) always underperforms its context-aware counterpart. However, even the best performing models lag well behind humans.

Image-text: We combine the previous two models by concatenating the output of the intrapanel LSTM with the `fc7` representation of the image and passing the result through a fully-connected layer before feeding it to the interpanel LSTM (Figure 6.7). For *text cloze* and *character coherence*, we also experiment with a variant of the image-text baseline that has no access to the context panels, which we dub **NC-image-text**. In this model, the scoring function computes inner products between the image features of p_i and the text candidates.[17]

6.8 Error Analysis

Table 4.4.2 contains our full experimental results, which we briefly summarize here. On *text cloze*, the image-text model dominates those trained on a single modality. However, text is much less helpful for *visual cloze* than it is for *text cloze*, suggesting that visual similarity dominates the former task. Having the context of the preceding panels helps

[17] We cannot apply this model to *visual cloze* because we are not allowed access to the artwork in panel p_i.

across the board, although the improvements are lower in the *hard* setting. There is more variation across the models in the *easy* setting; we hypothesize that the *hard* case requires moving away from pretrained image features, and transfer learning methods may prove effective here. Differences between models on *character coherence* are minor; we suspect that more complicated attentional architectures that leverage the bounding box locations b_{i_x} are necessary to "follow" speech bubble tails to the characters who speak them.

We also compare all models to a human baseline, for which the authors manually solve one hundred instances of each task (in the *hard* setting) given the same preprocessed input that is fed to the neural architectures. Most human errors are the result of poor OCR quality (e.g., misspelled words) or low image resolution. Humans comfortably outperform all models, making it worthwhile to look at where computers fail but humans succeed.

Figure 6.9: Three *text cloze* examples from the development set, shown with a single panel of context (boxed candidates are predictions by the **text-image** model). The airplane artwork in the top row helps the **image-text** model choose the correct answer, while the **text-only** model fails because the dialogue lacks contextual information. Conversely, the bottom two rows show the **image-text** model ignoring the context in favor of choosing a candidate that mentions something visually present in the last panel.

The top row in Figure 6.8 demonstrates an instance (from *easy text cloze* where the image helps the model make the correct prediction. The text-only model has no idea that an airplane (referred to here as a "ship") is present in the panel sequence, as the dialogue in the context panels make no mention of it. In contrast, the image-text model is able to use the artwork to rule out the two incorrect candidates.

The bottom two rows in Figure 6.8 show *hard text cloze* instances in which the image-text model is deceived by the artwork in the final panel. While the final panel of the middle row does contain what looks to be a creek, "catfish creek jail" is more suited for a narrative box than a speech bubble, while the meaning of the correct candidate is obscured by the dialect and out-of-vocabulary token. Similarly, a camera films a fight scene in the last row; the model selects a candidate that describes a fight instead of focusing on the context in which the scene occurs. These examples suggest that the contextual information is overridden by strong associations between text and image, motivating architectures that go beyond similarity by leveraging external world knowledge to determine whether an utterance is truly appropriate in a given situation.

6.9 Related Work

Our work is related to three main areas: (1) multimodal tasks that require language and vision understanding, (2) computational methods that focus on non-natural images, and (3) models that characterize language-based narratives.

Deep learning has renewed interest in jointly reasoning about vision and language. Datasets such as MS COCO (Lin et al., 2014) and Visual Genome (Krishna et al., 2016)

have enabled image captioning (Vinyals et al., 2015; Karpathy and Li, 2015; Xu et al., 2015) and visual question answering (Malinowski et al., 2015; Lu et al., 2016). Similar to our *character coherence* task, researchers have built models that match TV show characters with their visual attributes (Everingham et al., 2006) and speech patterns (Haurilet et al., 2016).

Closest to our own comic book setting is the visual storytelling task, in which systems must generate (Huang et al., 2016a) or reorder (Agrawal et al., 2016) stories given a dataset (SIND) of photos from Flikr galleries of "storyable" events such as weddings and birthday parties. SIND's images are fundamentally different from **COMICS** in that they lack coherent characters and accompanying dialogue. Comics are created by skilled professionals, not crowdsourced workers, and they offer a far greater variety of character-centric stories that depend on dialogue to further the narrative; with that said, the text in **COMICS** is less suited for generation because of OCR errors.

We build here on previous work that attempts to understand non-natural images. Zitnick et al. (Zitnick et al., 2016) discover semantic scene properties from a clip art dataset featuring characters and objects in a limited variety of settings. Applications of deep learning to paintings include tasks such as detecting objects in oil paintings (Crowley and Zisserman, 2014; Crowley et al., 2015) and answering questions about artwork (Guha et al., 2016). Previous computational work on comics focuses primarily on extracting elements such as panels and textboxes (Rigaud, 2014); in addition to the references in Section 6.2, there is a large body of segmentation research on manga (Aramaki et al., 2014; Pang et al., 2014a; Matsui, 2015; Kovanen and Aizawa, 2015).

To the best of our knowledge, we are the first to computationally model *content* in

comic books as opposed to just extracting their elements. We follow previous work in language-based narrative understanding; very similar to our *text cloze* task is the "Story Cloze Test" (Mostafazadeh et al., 2016), in which models must predict the ending to a short (four sentences long) story. Just like our tasks, the Story Cloze Test proves difficult for computers and motivates future research into commonsense knowledge acquisition. Others have studied characters (Elson et al., 2010; Bamman et al., 2014; Iyyer et al., 2016) and narrative structure (Schank and Abelson, 1977; Lehnert, 1981; Chambers and Jurafsky, 2009) in novels.

6.10 Conclusion & Future Work

We present the **COMICS** dataset, which contains over 1.2 million panels from "Golden Age" comic books. We design three cloze-style tasks on **COMICS** to explore *closure*, or how readers connect disparate panels into coherent stories. Experiments with different neural architectures, along with a manual data analysis, confirm the importance of multimodal models that combine text and image for comics understanding. We additionally show that context is crucial for predicting narrative or character-centric aspects of panels.

However, for computers to reach human performance, they will need to become better at leveraging context. Readers rely on commonsense knowledge to make sense of dramatic scene and camera changes; how can we inject such knowledge into our models? Another potentially intriguing direction, especially given recent advances in generative adversarial networks (Goodfellow et al., 2014), is generating artwork given dialogue (or vice versa). Finally, **COMICS** presents a golden opportunity for transfer learning; can we

train models that generalize across natural and non-natural images much like humans do?

Chapter 7

Conclusion

In this thesis, I have explored a variety of deep neural network architectures for tasks that involve both small and large-scale discourse-level understanding. These tasks encompass a wide variety of contexts, from short questions to sequences of images; the unifying factor is our ability to tackle all of them using a collection of neural network modules. The work here departs from traditional NLP work on discourse by not specifying the *type* of relations between different units of text as we would in rhetorical structure theory (see Section 2.5.1 for details), for example. Instead, neural networks implicitly reason about these connections; in the relationships modeling work of Chapter 5, we do not specify anything other than the number of relationship types, and the model fills them by leveraging patterns it learns from the input data. Deep learning holds much promise for discourse-level representation learning; to wrap up my thesis, I will first recap each chapter before offering proposed directions for future research in this area.

7.1 Contextual Question Answering

I first presented two question-answering tasks, quiz bowl and sequential semantic parsing, which focused on small paragraph-length inputs. For quiz bowl, we have complete supervision in the form of question-answer pairs, and simple vector averaging serves to effectively aggregate information across sentences. In contrast, our semantic parsing

setting is not fully supervised: we have question-answer pairs but lack the ground-truth intermediate logical form. This makes both answering questions and using information from previous questions more difficult than in quiz bowl; we settle on a modular neural network trained via structured output learning.

7.1.1 Future Directions for QA

For both tasks, we look at small context sizes, which was necessary to make the problems approachable. Here I outline future directions that seek to expand context complexity and size.

Quiz bowl Quiz bowl contains questions about language domains such as literature with huge, convoluted contexts (as we saw with the fictional relationships in Chapter 5). Simply training on paragraph-long questions is not enough to answer these questions at early positions, especially if the clues do not occur during training. For example, take this clue from a question on Henrik Ibsen's "A Doll's House": *In a scene from this play, one character practices a tarantella to prevent another character from opening his mail.* We rarely expect to see references to this particular scene during training, if at all, as it is relatively obscure. The only way to answer clues like this is to allow the model access to the raw source material, which opens up another can of worms: how do we map this clue to scenes in the play? This "inverse summarization" problem is a potentially very interesting avenue of research, as it requires both small and large-scale context understanding.

Sequential semantic parsing The SQA dataset is a first step in the direction of conversational QA. With that said, it is not by any means a perfect simulation of real conversation. We assume that the user is always asking a question at each turn, and that the computer is always answering it. This paradigm is of course not always true in real life, as computers may want to ask clarifying questions given underspecified queries, and users may want to incorporate more chat-like turns rather than bombarding the computer with questions in an effort to make the conversation seem more natural. To bring the task closer to real-world conversation, we need to equip our QA models with the ability to *generate* language in addition to existing semantic parse functionality. This is a challenging goal because not only does the network need to generate grammatical, meaningful utterances, but it also needs to decide when to switch to "chat mode" and when to execute a semantic parse over a knowledge base.

7.2 Comprehending Novels and Comics

The most difficult problems I tackle in this thesis are applications of deep learning to creative domains. In the QA tasks, we looked at short contexts in a relatively small answer space; for quiz bowl, we have a fixed set of a few thousand answers, while for SQA each question's answer space is defined by its corresponding table. In Chapter 5, I propose a neural network architecture to model the dynamics of fictional relationships that span entire novels, while in Chapter 6 I explore comic narrative understanding with deep learning.

7.2.1 Future Directions in Creative Understanding

There is a long (and perhaps impossible) road ahead for machines before they can "read" a novel or comic book at the same level of understanding as human readers. For one, humans possess a wealth of world knowledge and personal experiences that they can access while reading, while neural networks start with a blank slate and have to pick up world knowledge and commonsense reasoning just from their training data. An important open question is exactly how to bake world knowledge into these networks prior to or during training; can purely unsupervised methods learn this knowledge from raw text, or do we need to leverage large annotated resources?

Regarding the relationship modeling task, there are many avenues for further research. For example, the **RMN** model ignores asymmetric relationships, and so modeling unrequited love and other more complex relationships is not feasible within the current framework. To capture these sorts of relationships, we cannot use bag-of-words models like the **DAN** to compose span representations, as syntactic features are crucial to determine agent-patient relationships and other features indicative of asymmetry; architectures such as **TreeNN**s could perhaps be of value here. Another potential future direction is considering the entire book rather than just spans of text that contain relevant character mentions, as interactions between other characters and descriptive language about the environment are useful sources of information.

Finally, in addition to the future directions mentioned in Chapter 6 regarding the comic books project, there are more ambitious tasks to be solved in this domain. Generating dialogue and artwork using adversarial networks is one such application. A less lofty goal

might be to accomplish panel reordering: given a set of n panels from a comic book page, can a model learn to sort them into the correct order? To solve this task for large values of n, the model needs a lot of external knowledge; pretraining on frames from movies is a concrete way to inject such knowledge into the network.

Bibliography

Alekh Agarwal, Animashree Anandkumar, Prateek Jain, Praneeth Netrapalli, and Rashish Tandon. Learning sparsely used overcomplete dictionaries. In *Proceedings of Conference on Learning Theory*, 2014.

Harsh Agrawal, Arjun Chandrasekaran, Dhruv Batra, Devi Parikh, and Mohit Bansal. Sort story: Sorting jumbled images and captions into stories. In *Proceedings of Empirical Methods in Natural Language Processing*, 2016.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Learning to compose neural networks for question answering. In *Conference of the North American Chapter of the Association for Computational Linguistics*, 2016.

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. VQA: Visual question answering. In *International Conference on Computer Vision*, 2015.

Yuji Aramaki, Yusuke Matsui, Toshihiko Yamasaki, and Kiyoharu Aizawa. Interactive segmentation for manga. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference*, 2014.

Yoav Artzi and Luke Zettlemoyer. Bootstrapping semantic parsers from conversations. In *Proceedings of Empirical Methods in Natural Language Processing*, 2011.

Yusuf Aytar, Lluis Castrejon, Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Cross-modal scene networks. *arXiv*, 2016. URL http://arxiv.org/abs/1610.09003.

David Bamman, Brendan O'Connor, and Noah A. Smith. Learning latent personas of film characters. In *Proceedings of the Association for Computational Linguistics*, 2013.

David Bamman, Ted Underwood, and Noah A. Smith. A Bayesian mixed effects model of literary character. In *Proceedings of the Association for Computational Linguistics*, 2014.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

Taylor Berg-Kirkpatrick, Greg Durrett, and Dan Klein. Unsupervised transcription of historical documents. In *Proceedings of the Association for Computational Linguistics*, 2013.

David M Blei, Andrew Y Ng, and Michael I Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3, 2003.

Jordan Boyd-Graber, Brianna Satinoff, He He, and Hal Daumé III. Besting the quiz master: Crowdsourcing incremental classification games. In *Proceedings of Empirical Methods in Natural Language Processing*, 2012.

Ziqiang Cao, Sujian Li, Yang Liu, Wenjie Li, and Heng Ji. A novel neural topic model and its supervised extension. In *Association for the Advancement of Artificial Intelligence*, 2015.

Lynn Carlson, Daniel Marcu, and Mary Ellen Okurowski. Building a discourse-tagged corpus in the framework of rhetorical structure theory. In *Current and new directions in discourse and dialogue*, 2003.

Patrick J Carroll, Jason R Young, and Michael S Guertin. Visual analysis of cartoons: A view from the far side. In *Eye movements and visual cognition*. Springer, 1992.

Nathanael Chambers and Dan Jurafsky. Unsupervised learning of narrative schemas and their participants. In *Proceedings of the Association for Computational Linguistics*, 2009.

Nathanael Chambers and Daniel Jurafsky. Unsupervised learning of narrative event chains. In *Proceedings of the Association for Computational Linguistics*, 2008.

Jonathan Chang, Jordan Boyd-Graber, and David M Blei. Connections between the lines: augmenting social networks with text. In *Knowledge Discovery and Data Mining*, 2009a.

Jonathan Chang, Sean Gerrish, Chong Wang, Jordan L Boyd-Graber, and David M Blei. Reading tea leaves: How humans interpret topic models. In *Proceedings of Advances in Neural Information Processing Systems*, 2009b.

Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *British Machine Vision Conference*, 2014.

Snigdha Chaturvedi, Shashank Srivastava, Hal Daumé III, and Chris Dyer. Modeling dynamic relationships between characters in literary novels. In *Association for the Advancement of Artificial Intelligence*, 2016.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Proceedings of Empirical Methods in Natural Language Processing*, 2014.

J. Cognard-Black, M. Goldthwaite, and M. Nestle. *Books That Cook: The Making of a Literary Meal*. NYU Press, 2014. ISBN 9781479830213. URL https://books.google.com/books?id=5IETCgAAQBAJ.

Neil Cohn. The limits of time and transitions: challenges to theories of sequential image comprehension. *Studies in Comics*, 1(1), 2010.

E. J. Crowley, O. M. Parkhi, and A. Zisserman. Face painting: querying art with photos. In *British Machine Vision Conference*, 2015.

Elliot Crowley and Andrew Zisserman. The state of the art: Object retrieval in paintings using discriminative regions. In *British Machine Vision Conference*, 2014.

Rajarshi Das, Manzil Zaheer, and Chris Dyer. Gaussian lda for topic models with word embeddings. In *Proceedings of the Association for Computational Linguistics*, 2015.

Bhuwan Dhingra, Hanxiao Liu, Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Gated-attention readers for text comprehension. In *Proceedings of the Association for Computational Linguistics*, 2017.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 2011.

Will Eisner. *Comics & Sequential Art*. Poorhouse Press, 1990.

Michael Elad and Michal Aharon. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Transactions on Image Processing*, 15(12), 2006.

Micha Elsner. Character-based kernels for novelistic plot structure. In *Proceedings of the European Chapter of the Association for Computational Linguistics*, 2012.

David K Elson, Nicholas Dames, and Kathleen R McKeown. Extracting social networks from literary fiction. In *Proceedings of the Association for Computational Linguistics*, 2010.

Mark Everingham, Josef Sivic, and Andrew Zisserman. Hello! my name is... Buffy" – automatic naming of characters in TV video. In *Proceedings of the British Machine Vision Conference*, 2006.

Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. Open question answering over curated and extracted knowledge bases. In *Knowledge Discovery and Data Mining*, 2014.

Manaal Faruqui, Yulia Tsvetkov, Dani Yogatama, Chris Dyer, and Noah Smith. Sparse overcomplete word vector representations. In *Proceedings of the Association for Computational Linguistics*, 2015.

Vanessa Wei Feng and Graeme Hirst. Text-level discourse parsing with rich linguistic features. In *Proceedings of the Association for Computational Linguistics*, 2012.

Joseph L Fleiss. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5), 1971.

Lucie Flekova and Iryna Gurevych. Personality profiling of fictional characters using sense-level links between lexical resources. In *Proceedings of Empirical Methods in Natural Language Processing*, 2015.

T.C. Foster. *How to Read Literature Like a Professor*. HarperCollins, 2009. ISBN 9780061804069. URL https://books.google.com/books?id=PKeOyeZdVKwC.

Tom Foulsham, Dean Wybrow, and Neil Cohn. Reading without words: Eye movements in the comprehension of comic strips. *Applied Cognitive Psychology*, 30, 2016.

Stefan L Frank. Uncertainty reduction as a measure of cognitive load in sentence comprehension. *Topics in Cognitive Science*, 5(3), 2013.

Alona Fyshe, Leila Wehbe, Partha P Talukdar, Brian Murphy, and Tom M Mitchell. A compositional and interpretable semantic space. In *Conference of the North American Chapter of the Association for Computational Linguistics*, 2015.

Christoph Goller and Andreas Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In *Neural Networks, 1996., IEEE International Conference on*, volume 1, 1996.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proceedings of Advances in Neural Information Processing Systems*, 2014.

Michael D Gordin, Helen Tilley, and Gyan Prakash. *Utopia/dystopia: conditions of historical possibility*. Princeton University Press, 2010.

Ron Goulart. *Comic Book Encyclopedia: The Ultimate Guide to Characters, Graphic Novels, Writers, and Artists in the Comic Book Universe*. HarperCollins, 2004.

Amit Goyal, Ellen Riloff, and Hal Daumé III. A computational model for plot units. *Computational Intelligence Journal*, 29(3), 2013.

Amit Gruber, Yair Weiss, and Michal Rosen-Zvi. Hidden topic markov models. In *Proceedings of Artificial Intelligence and Statistics*, 2007.

Clément Guérin, Christophe Rigaud, Antoine Mercier, Farid Ammar-Boudjelal, Karell Bertet, Alain Bouju, Jean-Christophe Burie, Georges Louis, Jean-Marc Ogier, and Arnaud Revel. eBDtheque: A representative database of comics. In *International Conference on Document Analysis and Recognition*, 2013.

Anupam Guha, Mohit Iyyer, and Jordan Boyd-Graber. A distorted skull lies in the bottom center: Identifying paintings from text descriptions. In *NAACL Human-Computer Question Answering Workshop*, 2016.

John Hale. Uncertainty about the rest of the sentence. *Cognitive Science*, 30(4), 2006.

Michael Halliday and Ruqaiya Hasan. *Cohesion in English*. 1976.

Sanda Harabagiu, Andrew Hickl, John Lehmann, and Dan Moldovan. Experiments with interactive question-answering. In *Proceedings of the Association for Computational Linguistics*, 2005.

Monica-Laura Haurilet, Makarand Tapaswi, Ziad Al-Halah, and Rainer Stiefelhagen. Naming TV characters by watching and analyzing dialogs. In *IEEE Winter Conference on Applications of Computer Vision*, 2016.

Marti A Hearst. Texttiling: Segmenting text into multi-paragraph subtopic passages. *Computational linguistics*, 23(1), 1997.

Karl Moritz Hermann, Edward Grefenstette, and Phil Blunsom. "not not bad" is not "bad": A distributional account of negation. *Proceedings of the ACL Workshop on Continuous Vector Space Models and their Compositionality*, 2013.

Karl Moritz Hermann, Dipanjan Das, Jason Weston, and Kuzman Ganchev. Semantic frame identification with distributed word representations. In *Proceedings of the Association for Computational Linguistics*, 2014.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

Anh Khoi Ngo Ho, Jean-Christophe Burie, and Jean-Marc Ogier. Panel and speech balloon extraction from comic books. In *IAPR International Workshop on Document Analysis Systems*, 2012.

Jerry R Hobbs. Coherence and coreference. *Cognitive science*, 3(1), 1979.

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

Ting-Hao Kenneth Huang, Francis Ferraro, Nasrin Mostafazadeh, Ishan Misra, Aishwarya Agrawal, Jacob Devlin, Ross Girshick, Xiaodong He, Pushmeet Kohli, Dhruv Batra, et al. Visual storytelling. In *Conference of the North American Chapter of the Association for Computational Linguistics*, 2016a.

Ting-Hao (Kenneth) Huang, Francis Ferraro, Nasrin Mostafazadeh, Ishan Misra, Aishwarya Agrawal, Jacob Devlin, Ross B. Girshick, Xiaodong He, Pushmeet Kohli, Dhruv Batra, C. Lawrence Zitnick, Devi Parikh, Lucy Vanderwende, Michel Galley, and Margaret Mitchell. Visual storytelling. In *Conference of the North American Chapter of the Association for Computational Linguistics*, 2016b.

Aapo Hyvärinen and Erkki Oja. Independent component analysis: algorithms and applications. *Neural networks*, 13(4), 2000.

Mohit Iyyer, Jordan Boyd-Graber, Leonardo Claudino, Richard Socher, and Hal Daumé III. A neural network for factoid question answering over paragraphs. In *Proceedings of Empirical Methods in Natural Language Processing*, 2014a.

Mohit Iyyer, Peter Enns, Jordan Boyd-Graber, and Philip Resnik. Political ideology detection using recursive neural networks. In *Proceedings of the Association for Computational Linguistics*, 2014b.

Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the Association for Computational Linguistics*, 2015.

Mohit Iyyer, Anupam Guha, Snigdha Chaturvedi, Jordan Boyd-Graber, and Hal Daumé III. Feuding families and former friends: Unsupervised learning for dynamic fictional relationships. In *Conference of the North American Chapter of the Association for Computational Linguistics*, 2016.

Mohit Iyyer, Varun Manjunatha, Anupam Guha, Yogarshi Vyas, Jordan Boyd-Graber, Hal Daumé III, and Larry Davis. The amazing mysteries of the gutter: Drawing inferences between panels in comic book narratives. In *Computer Vision and Pattern Recognition*, 2017a.

Mohit Iyyer, Wen tau Yih, and Ming-Wei Chang. Search-based neural structured learning for sequential question answering. In *Proceedings of the Association for Computational Linguistics*, 2017b.

Max Jaderberg, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Reading text in the wild with convolutional neural networks. *International Journal of Computer Vision*, 116(1), 2016.

Yangfeng Ji and Jacob Eisenstein. Representation learning for text-level discourse parsing. In *Proceedings of the Association for Computational Linguistics*, 2014.

Matt L. Jockers. *Macroanalysis: Digital Methods and Literary History*. Topics in the Digital Humanities. University of Illinois Press, 2013. ISBN 9780252094767. URL http://books.google.com/books?id=mPOdxQgpOSUC.

Andrej Karpathy and Fei-Fei Li. Deep visual-semantic alignments for generating image descriptions. In *Computer Vision and Pattern Recognition*, 2015.

Diane Kelly and Jimmy Lin. Overview of the trec 2006 ciQA task. In *ACM SIGIR Forum*, volume 41, 2007.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*, 2014.

Samu Kovanen and Kiyoharu Aizawa. A layered method for determining manga text bubble reading order. In *International Conference on Image Processing*, 2015.

Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, Michael Bernstein, and Li Fei-Fei. Visual genome: Connecting language and vision using crowdsourced dense image annotations. In *International Journal of Computer Vision*, 2016.

Vinodh Krishnan and Jacob Eisenstein. "You're Mr. Lebowski, I'm The Dude": Inducing address term formality in signed social networks. In *Conference of the North American Chapter of the Association for Computational Linguistics*, 2015.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of Advances in Neural Information Processing Systems*, 2012.

Angeliki Lazaridou, Elia Bruni, and Marco Baroni. Is this a wampimuk? Cross-modal mapping between distributional semantics and the visual world. In *Proceedings of the Association for Computational Linguistics*, 2014.

Quoc V Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the International Conference of Machine Learning*, 2014.

Wendy G Lehnert. Plot units and narrative summarization. *Cognitive Science*, 5(4), 1981.

Roger Levy. Expectation-based syntactic comprehension. *Cognition*, 106(3), 2008.

Jiwei Li, Minh-Thang Luong, and Dan Jurafsky. A hierarchical neural autoencoder for paragraphs and documents. In *Proceedings of the Association for Computational Linguistics*, 2015.

Luyuan Li, Yongtao Wang, Zhi Tang, and Liangcai Gao. Automatic comic page segmentation based on polygon detection. *Multimedia Tools and Applications*, 69(1), 2014.

Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao. Neural symbolic machines: Learning semantic parsers on Freebase with weak supervision. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017.

Percy Liang. Learning executable semantic parsers for natural language understanding. *Communications of the ACM*, 59(9), 2016.

Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *Proceedings of the European Conference on Computer Vision*, 2014.

Reginald Long, Panupong Pasupat, and Percy Liang. Simpler context-dependent logical forms via model projections. In *Proceedings of the Association for Computational Linguistics*, 2016.

Moshe Looks, Marcello Herreshoff, DeLesley Hutchins, and Peter Norvig. Deep learning with dynamic computation graphs. In *Proceedings of the International Conference on Learning Representations*, 2017.

Jiasen Lu, Jianwei Yang, Dhruv Batra, and Devi Parikh. Hierarchical question-image co-attention for visual question answering, 2016.

Mateusz Malinowski, Marcus Rohrbach, and Mario Fritz. Ask your neurons: A neural-based approach to answering questions about images. In *Computer Vision and Pattern Recognition*, 2015.

William C Mann and Sandra A Thompson. Rhetorical structure theory: Toward a functional theory of text organization. *Text-Interdisciplinary Journal for the Study of Discourse*, 8, 1988.

Philip Massey, Patrick Xia, David Bamman, and Noah A Smith. Annotating character relationships in literary texts. *arXiv:1512.00728*, 2015.

Yusuke Matsui. Challenge for manga processing: Sketch-based manga retrieval. In *Proceedings of the 23rd Annual ACM Conference on Multimedia*, 2015.

Yusuke Matsui, Kota Ito, Yuji Aramaki, Toshihiko Yamasaki, and Kiyoharu Aizawa. Sketch-based manga retrieval using manga109 dataset. *arXiv preprint arXiv:1510.04389*, 2015.

Jon D Mcauliffe and David M Blei. Supervised topic models. In *Proceedings of Advances in Neural Information Processing Systems*, 2008.

Scott McCloud. *Understanding Comics*. HarperCollins, 1994.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Eleni Miltsakaki, Rashmi Prasad, Aravind K Joshi, and Bonnie L Webber. The penn discourse treebank. In *International Language Resources and Evaluation*, 2004.

Guy M Morton. *A computer oriented geodetic data base and a new technique in file sequencing*. International Business Machines Co, 1966.

Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. A corpus and cloze evaluation for deeper understanding of commonsense stories. In *Conference of the North American Chapter of the Association for Computational Linguistics*, 2016.

Arvind Neelakantan, Quoc Le, Martin Abadi, Andrew McCallum, and Dario Amodei. Learning a natural language interface with neural programmer. In *Proceedings of the International Conference on Learning Representations*, 2017.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. DyNet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*, 2017.

Viet-An Nguyen, Jordan Boyd-Graber, Philip Resnik, Deborah Cai, Jennifer Midberry, and Yuanxin Wang. Modeling topic control to detect influence in conversations using nonparametric topic models. *Machine Learning*, 95, 2014.

Bruno A Olshausen and David J Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 37(23), 1997.

J Walker Orr, Prasad Tadepalli, Janardhan Rao Doppa, Xiaoli Fern, and Thomas G Dietterich. Learning scripts as hidden markov models. In *Association for the Advancement of Artificial Intelligence*, 2014.

Mark Palatucci, Dean Pomerleau, Geoffrey E. Hinton, and Tom M. Mitchell. Zero-shot learning with semantic output codes. In *Proceedings of Advances in Neural Information Processing Systems*, 2009.

Xufang Pang, Ying Cao, Rynson W. H. Lau, and Antoni B. Chan. A robust panel extraction method for manga. In *Proceedings of the ACM International Conference on Multimedia*, 2014a.

Xufang Pang, Ying Cao, Rynson WH Lau, and Antoni B Chan. A robust panel extraction method for manga. In *Proceedings of the ACM International Conference on Multimedia*, 2014b.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the International Conference of Machine Learning*, 2013.

Rebecca J Passonneau and Diane J Litman. Discourse segmentation by human and automated means. *Computational Linguistics*, 23(1), 1997.

P. Pasupat and P. Liang. Zero-shot entity extraction from web pages. In *Proceedings of the Association for Computational Linguistics*, 2014.

Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. In *Proceedings of the Association for Computational Linguistics*, 2015.

Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of Empirical Methods in Natural Language Processing*, 2014.

Jordan B Pollack. Recursive distributed representations. *Artificial Intelligence*, 1990.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Proceedings of Advances in Neural Information Processing Systems*, 2015.

Christophe Rigaud. *Segmentation and indexation of complex objects in comic book images*. PhD thesis, University of La Rochelle, France, 2014.

Christophe Rigaud, Jean-Christophe Burie, Jean-Marc Ogier, Dimosthenis Karatzas, and Joost Van de Weijer. An active contour model for speech balloon detection in comics. In *International Conference on Document Analysis and Recognition*, 2013.

Christophe Rigaud, Clément Guérin, Dimosthenis Karatzas, Jean-Christophe Burie, and Jean-Marc Ogier. Knowledge-driven understanding of images in comic books. *International Journal on Document Analysis and Recognition*, 18(3), 2015.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1*, 1986.

Gerard Salton and Michael J McGill. Introduction to modern information retrieval. 1986.

Roger Schank and Robert Abelson. *Scripts, Plans, Goals and Understanding: An Inquiry into Human Knowledge Structures*. L. Erlbaum, 1977.

Ray Smith. An overview of the tesseract ocr engine. In *International Conference on Document Analysis and Recognition*, 2007.

Richard Socher, Eric H. Huang, Jeffrey Pennington, Andrew Y. Ng, and Christopher D. Manning. Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection. In *Proceedings of Advances in Neural Information Processing Systems*, 2011a.

Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions. In *Proceedings of Empirical Methods in Natural Language Processing*, 2011b.

Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of Empirical Methods in Natural Language Processing*, 2013.

Richard Socher, Quoc V Le, Christopher D Manning, and Andrew Y Ng. Grounded compositional semantics for finding and describing images with sentences. *Transactions of the Association for Computational Linguistics*, 2014.

Radu Soricut and Daniel Marcu. Sentence level discourse parsing using syntactic and lexical information. In *Conference of the North American Chapter of the Association for Computational Linguistics*, 2003.

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

Shashank Srivastava, Snigdha Chaturvedi, and Tom Mitchell. Inferring interpersonal relations in narrative summaries. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, 2016.

Edmund M. Talley, David Newman, David Mimno, Bruce W. Herr, Hanna M. Wallach, Gully A. P. C. Burns, A. G. Miriam Leenders, and Andrew McCallum. Database of NIH grants using machine-learned categories and graphical clustering. *Nature Methods*, 8(6), 2011.

Takamasa Tanaka, Kenji Shoji, Fubito Toyama, and Juichi Miyamichi. Layout analysis of tree-structured scene frames in comic images. In *International Joint Conference on Artificial Intelligence*, 2007.

Duyu Tang, Bing Qin, and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of Empirical Methods in Natural Language Processing*, 2015.

Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin Markov networks. In *Proceedings of Advances in Neural Information Processing Systems*, 2004.

Wilson L Taylor. Cloze procedure: a new tool for measuring readability. *Journalism and Mass Communication Quarterly*, 30(4), 1953.

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

Nicolas Usunier, David Buffoni, and Patrick Gallinari. Ranking with ordered weighted pairwise classification. In *Proceedings of the International Conference of Machine Learning*, 2009.

Hardik Vala, David Jurgens, Andrew Piper, and Derek Ruths. Mr. bennet, his coachman, and the archbishop walk into a bar but only one of them gets recognized: On the difficulty of detecting characters in literary texts. In *Proceedings of Empirical Methods in Natural Language Processing*, 2015.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 2008.

Karen van Hoek. *Anaphora and Conceptual Structure*. Cognitive Theory of Language and Culture Series. University of Chicago Press, 1997.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Computer Vision and Pattern Recognition*, 2015.

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78 (10), 1990.

Jason Weston, Samy Bengio, and Nicolas Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *International Joint Conference on Artificial Intelligence*, 2011.

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

Caiming Xiong, Stephen Merity, and Richard Socher. Dynamic memory networks for visual and textual question answering. In *Proceedings of the International Conference of Machine Learning*, 2016.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the International Conference of Machine Learning*, 2015.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1321–1331, July 2015.

Wen-tau Yih, Matthew Richardson, Chris Meek, Ming-Wei Chang, and Jina Suh. The value of semantic parse labeling for knowledge base question answering. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 201–206, Berlin, Germany, August 2016.

Ilinca Zarifopol-Johnston. *To kill a text: the dialogic fiction of Hugo, Dickens, and Zola*. University of Delaware Press, 1995.

Luke S Zettlemoyer and Michael Collins. Learning context-dependent mappings from sentences to logical form. In *Proceedings of the Association for Computational Linguistics*, 2009.

Bolei Zhou, Yuandong Tian, Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Simple baseline for visual question answering. *arXiv preprint arXiv:1512.02167*, 2015.

C. Lawrence Zitnick, Ramakrishna Vedantam, and Devi Parikh. Adopting abstract images for semantic scene understanding. *IEEE Trans. Pattern Anal. Mach. Intell.*, 38(4):627–638, 2016.