Structured State Tracking for Natural Language Understanding

A dissertation presented
by
Justin T Chiu
to
The Department of Cornell

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

Cornell University
New York City, New York
August 2024

Thesis advisor: Professor Alexander M Rush Justin T Chiu

Structured State Tracking for Natural Language Understanding

Abstract

Autonomous agents that collaborate with humans must understand language, track the state of the world, and make good decisions. A central challenge common to these three desiderata is uncertainty. Language is often ambiguous, with many possible interpretations of the same utterance. The state of the world is unobservable, as a single agent cannot observe every aspect due to physical constraints. Agents must reason about the unobserved aspects as new information is received. Finally, many decisions have uncertain outcomes. Agents must anticipate how the world will respond to their decisions in order to achieve the best outcomes.

In this thesis, we build uncertainty-aware agents that understand language, the state of the world, and decision-making. The thesis is divided into three sections, with each section exploring a different application by applying a modern twist to a classical approach. The first section studies uncertainty-aware representations in language modeling. We revisit classical structured and uncertainty-aware language models, showing that they can achieve strong performance when combined with modern techniques from deep learning. The second section studies the use of language models for reasoning about uncertainty in question-answering. We reason about unobserved reasoning paths, applying modern pretrained language model representations to aid in high-level reasoning. The third section studies task-oriented dialogue as decision-making under uncertainty. We intertwine classical decision-making methods with the powerful language and code understanding capabilities of modern large language models, resulting in an accurate neurosymbolic dialogue system.

Biographical Sketch

Justin Chiu completed his B.S. and M.S. degrees in Computer Engineering and Science from the University of Pennsylvania in 2015, served as a Research Engineer at Facebook AI Research lab for two years, started his Ph.D. degree in Computer Science at Harvard University in 2017, and completed the degree at Cornell University in 2024.

For my parents.

Acknowledgments

I am deeply grateful to my advisor, Sasha Rush, for guiding me through my doctoral studies. He has taught me how to communicate and think, and given me the tools to conduct science for the rest of my life.

I would like to give special thanks to: Wenting Zhao, for renewing my interest in natural language processing and passion for research; Derek Chen, for patiently teaching me how to organize; Daniel Fried and Nicholas Tomlin, for introducing me to the wonderful world of strategic interaction; and Mitchell Stern, for graciously connecting me with Berkeley NLP.

I am also incredibly grateful to all my collaborators and friends: Cornell labmates Jack Morris, Woojeong Kim, Celine Lee and Nathan Yan; Cornell Tech NLP neighbours Alane Suhr, Ge Gao, Noriyuki Kojima, Anne Wu, Mustafa Omer Gul, Anya Ji, and Yoav Artzi; Cornell Tech ML neighbours Yair Schiff, Aaron Gokaslan, Subham Sahoo, Marianne Arriola, Jacqueline Maasch, Volodymyr Kuleshov, Evgenii Nikishin, and Angela Zhou; Saujas Vaduguru and Sedrick Keh at CMU; Nikita Kitaev, Steven Cao, and Dan Klein at Berkeley; Harvard labmates Yuntian Deng, Jiawei Zhou, Yoon Kim, Sam Wiseman, Sebastian Gehrmann, Allen Schmaltz, Zachary Ziegler, Kelly Zhang, Rachit Singh, Demi Guo, and Mirac Suzgun; Wendy Shang, Allan Jabri, Adam Fisch, Christian Puhrsch, Zeming Lin, Michael Auli, Armand Joulin, and Tomas Mikolov at Facebook; and Aarti Bagul, Wojciech Zaremba, Uri Shalit, and David Sontag at NYU.

Finally, I could not have done this without my family. Sheri Chiu for being kind and generous; Chi Chiu for being reliable; Godwin Chang for the poignant advice; and Susan Chang for having faith in me and giving me immeasurable support. Most importantly, I cannot express enough gratitude towards Kathy Choy, without whose unwavering support and delicious meals I would not be here.

Contents

Listing of figures

1

Introduction

A goal of artificial intelligence is to create autonomous problem-solving agents that can operate in the world and make effective decisions (Russell & Norvig, 2010). For this to happen, autonomous agents need two capabilities: They must represent the current knowledge of the world and then search for the optimal decision. These two facets of problem solving, state representation and search, are both deeply intertwined and problem-dependent. Much research has gone into problem-specific representations, demonstrating how they can improve search (Pearl, 1984, Young, 2006, Williams et al., 2013, Silver et al., 2017). In particular, informative state representations often make search easier, allowing methods to achieve a strong accuracy-speed tradeoff relative to more expensive search methods (Yang et al., 2021a, Ruoss et al., 2024).

State tracking is the use of state representations to model the world over time. Historically, state tracking originated from the fields of dynamical system and control theory, where the goal is to make optimal decisions by maintaining and planning through an accurate and updatable representation of the world. State tracking also plays a key role in machine learning, artificial intelligence, and natural language processing. An example of state tracking in machine learning is recurrent neural network for sequence modeling, which must compose observations to update their internal, learned state representations in order to predict the next observation (Elman, 1990, Hochreiter & Schmidhuber, 1997). In artificial intelligence, search algorithms must perform state tracking. For example, agents for multi-player games such as Hanabi or Diplomacy must track the state of the game in addition to beliefs over the unobserved state – such as the intentions of other players (Bard et al., 2020, FAIR et al., 2022). The applications of state tracking in natural language processing mirror those in machine learning and artificial intelligence. There are both sequence modeling and search problems: Language modeling is a sequence modeling problem, while task-oriented dialogue is a search problem. In task-oriented dialogue, the state of the world includes both the agent's environment and their human partner. An example is a flight-booking agent, where the agent helps the human partner decide on the destination, dates, airline, and other details. Throughout a flight-booking conversation, the agent must track the information gathered from the partner and elicit the information needed next.

This thesis focuses on state tracking representations for natural language understanding. We have two goals in designing state-tracking representations. Our first goal is explicitly model uncertainty. Uncertainty is a necessary component of state tracking. Agents rarely have full knowledge of the world, yet must reason about unknowns in order to communicate with others and interact with their environment. We focus on discrete representations, which allow for simple formal representations of uncertainty. Our second goal is to study representations that can be adapted efficiently: Representations must be efficient to update both in terms of training and incorporating new information. Throughout the thesis, we target finite discrete representations, which are interpretable (Bottou & LeCun, 2005, Ross et al., 2021), expressive, and tractable (Choi et al., 2020).

We approach both representational goals, uncertainty and adaptability, through the framework of probabilistic modeling. Probabilistic modeling provides a declarative framework that allows practitioners to separate concerns regarding modeling how the data is generated and infer missing data from observations. Framing state tracking as probabilistic modeling allows us to formalize the trade-offs involved in uncertainty and adaptation. Viewed under the formalism of probabilistic modeling, uncertainty is a set of model constraints and adaptation is inference under a particular model. This thesis explores probabilistic state representations, varying the constraints on the state representations from compositional, discrete states (chapters 3 and 4) to language (chapter 5), as well as symbolic and executable representations (chapters 6 and 7).

We will explore discrete representations for three applications. The first application is language modeling (chapters 3 and 4). The goal of language modeling is to predict the next token given all previous tokens. For example, given 'A quick brown,' the next word is likely 'fox'. The goal of state tracking within language modeling is to efficiently represent and update the information from past words to predict future words. The second application is extractive question answering (chapter 5). The goal of question answering is to answer a question by selecting information from supporting documents and then generating the answer itself. For example, 'Who is the queen of England?' The state representation for question answering must track which information has been selected to efficiently select the next necessary piece of supporting evidence. The third application is task-oriented dialogue (chapter 7). The goal of task-oriented dialogue is for the agent to solve a task, such as a reference game, collaboratively with a partner. The state representation for dialogue must track what information has been gathered from the dialogue so far to decide what to say next.

1.1 Thesis outline

We begin with a background that introduces how to model state representations in the framework of latent variable models. Each subsequent chapter explores a different state representation, presenting a method for learning or utilizing the given state representation for decision-making.

- Chapter 2, Background: State Tracking Models, Parameterization, and Inference, introduces the background on latent variable models for state representations, how to parameterize them with neural networks, and how to perform inference.
- Chapter 3, Scaling Hidden Markov Language Models, proposes a scalable, discrete state representation for language modeling.
- Chapter 4, Low-Rank Constraints for Fast Inference in Structured Models, extends the discrete state representation in the previous chapter to hierarchical models for language modeling.
- Chapter 5, HOP, UNION, GENERATE: Explainable Multi-hop Reasoning without Rationale Supervision, explores a natural language state representation for

question answering.

- Chapter 6, Modeling Perspective-Dependent Ambiguity in Collaborative Dialogue, begins a symbolic, probabilistic state representation for information-seeking dialogue.

- Chapter 7, Symbolic Planning and Code Generation for Grounded Dialogue, completes the state representation for information-seeking dialogue by incorporating code generation for state tracking.

- Chapter 8 concludes the thesis and discusses the future of state tracking in NLP.

1.2 Related publications

Elements of this thesis are drawn from the following publications:

- Chiu, J. & Rush, A. (2020a). Scaling hidden Markov language models. In B. Webber, T. Cohn, Y. He, & Y. Liu (Eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1341–1349). Online: Association for Computational Linguistics.

- Chiu, J., Deng, Y., & Rush, A. (2021). Low-rank constraints for fast inference in structured models. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, & J. W. Vaughan (Eds.), *Advances in Neural Information Processing Systems*, volume 34 (pp. 2887–2898).: Curran Associates, Inc.

- Zhao, W., Chiu, J., Cardie, C., & Rush, A. (2023). Hop, union, generate: Explainable multi-hop reasoning without rationale supervision. In H. Bouamor, J. Pino, & K. Bali (Eds.), *Proceedings of the 2023 Conference on Empirical Methods*

in Natural Language Processing (pp. 16119–16130). Singapore: Association for Computational Linguistics.

- Chiu, J. T., Zhao, W., Fried, D., & Rush, A. M. (2022a). Modeling perspective-dependent ambiguity in collaborative dialogue. In *The Third Wordplay: When Language Meets Games Workshop*.

- Chiu, J., Zhao, W., Chen, D., Vaduguru, S., Rush, A., & Fried, D. (2023). Symbolic planning and code generation for grounded dialogue. In H. Bouamor, J. Pino, & K. Bali (Eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing* (pp. 7426–7436). Singapore: Association for Computational Linguistics.

2

Background: State Tracking Models, Parameterization, and Inference

2.1 Motivation

The goal of state tracking is to store past information in a concise representation for an autonomous agent to easily make future decisions. The design of state representations is guided primarily by what agents can be programmed or trained to understand. For example, past approaches to state tracking in the dialogue domain rely on hand-crafted representations that can be processed automatically by agents (Young, 2006, Thomson & Young, 2010). This thesis will explore a range of state representations, ranging from learned to hand-designed, as well as for a range of applications.

This chapter covers the background, presenting the unifying probabilistic framework through which we will study state tracking.

Probabilistic methods for state tracking afford practitioners principled methods to not only design the state representations but also control how they are updated given new information. Probabilistic state tracking methods decompose state representations into modular, interpretable components, then use standard probabilistic tools to specify, update, and potentially learn the state (Stratonovich, 1959, Thrun et al., 2005). While there is a rich history of probabilistic state tracking as partially observable Markov decision processes, our thesis focuses primarily on the representations, rather than the decision process itself.

2.2 Roadmap

All state tracking representations can be expressed under the framework of latent variable modeling. The goal of latent variable modeling is to discover the unobserved, compositional systems that govern how data is generated.

1. Section 2.3: We begin this background chapter by outlining the notation for probabilistic models.
2. Sections 2.4 and 2.5: We then introduce latent variable models, describe how they are decomposed into modular components, and show how those components can be used to answer questions.
3. Section 2.6: We explain how the conditional distributions in latent variable models can be parameterized by neural networks.

4. Section 2.7: We conclude with a brief introduction to large language models (LLMs) and how to use them.

2.3 Notation

We use x to denote the input, z the state representation, and y the output. Following the Bayesian modeling literature, the lowercase letters x, z, y denote both the random variables as well as the values (Gelman et al., 2004). The input consists of a sequence of T tokens $x = (x_1, x_2, \ldots, x_T)$, with each token from a finite vocabulary $x_t \in \mathcal{X}$. The state representation z is a discrete latent variable that takes values in $\mathcal{Z}$. We will explore different representations $\mathcal{Z}$: Integers (Chapters 3, 4, and 6), sequences of text (Chapter 5), symbolic (Chapter 6), and programs (Chapter 7). The output y will also vary depending on the application and will be either a single token in $\mathcal{Y}$ or a sequence of tokens $\mathcal{Y}^*$.

Distributions over random variables determine the relative rates at which events occur, and conditional distributions determine how the realizations of random variables correlate with each other. Models are joint distributions given by $p(y, z|x; \theta)$, with parameters $\theta \in \mathbb{R}^{\mathcal{X} \times \mathcal{Z} \times \mathcal{Y}}$, or $p(y, z|x)$ with parameters implicit. Joint distributions decompose into conditional distributions via the chain rule: $p(y|x, z)p(z|x)$.

We express common functionals of distributions as follows: Expectations are written as

$$\mathbb{E}_{p(y|x)}\left[f(y, x)\right] = \sum_y p(y|x) f(y, x).$$

The (conditional) entropy of a distribution measures how many bits are required to encode an output on average, and is the expected negative log probability of each event

$$H[p(y|x)] = -\sum_{x,y} p(y,x) \log p(y|x).$$

The Kullback-Leibler (KL) divergence measures how far one distribution is from another, and is the expected log odds ratio between the two distributions:

$$KL[p(y|x)||q(y|x)] = \sum_{x,y} p(x,y) \log \frac{p(y|x)}{q(y|x)}.$$

The expected information gain measures the expected reduction in uncertainty after making an observation, and is given by the expected change in information over z after observing x and y:

$$\mathbb{E}_{p(y|x)}\left[H[z] - H[z|x,y]\right].$$

We denote matrices $W \in \mathbb{R}^{n \times m}$ and vectors $b \in \mathbb{R}^n$. We index matrices and vectors via subscripts W_{ij} and b_i.

2.4 Latent Variable Models for State Tracking

We focus on models that encode inputs x into a state representation z, with an encoder model $p(z|x)$, then produce an output y based only on the information in z with the decoder model $p(y|z)$. Formally, we represent this by a probabilistic model with the joint distribution $p(y,z|x) = p(y|z)p(z|x)$.

This thesis covers three applications. We introduce and formulate each application as a state tracking instance below. For each application, we give the input, output, and state:

Example 2.4.1. Language modeling: The goal of language modeling is to predict the next word y given the input x: all previous words. An example input is $x = (\text{a}, \text{quick}, \text{brown})$, while the next word could be $y = \text{fox}$. In Chapters 3 and 4 we map the previous input words to integer state representations z, which serve to cluster inputs and outputs jointly. //

Example 2.4.2. Question answering: The goal of question answering is to predict the output answer y to an input question x. An example input is

$$x = (\text{Who is the queen of England?}, \text{The Queen has recently passed}\ldots),$$

where the first element is the question and the second element is a body of text where the answer may occur. In this case, the answer would be $y = \text{Not answerable}$. In chapter 5, we represent the state z as the subset of the context with which to find the answer. //

Example 2.4.3. Task-oriented dialogue: At a high level, task-oriented dialogue requires an agent to collaborate with their partner to achieve a given task. The goal of dialogue is to choose the next decision y, which may be an utterance or action, given the previous dialogue history as input x.

To make this concrete, consider the game of 20 questions. The agent's goal is to guess the latent target image out of a set of given images. The partner sees the set of images and knows which is the target. The agent gets 20 turns – in each turn, they can ask a single question such as 'Is it a giraffe?' The partner then answers the question, and so forth. In this setting, x is the set of all observed question and answer pairs, as well as the next question the agent plans to ask, the latent state representation z is the target image, and the partner's answer is y. //

Probabilistic models provide a declarative framework for specifying structure based on how the joint distribution is decomposed. For state tracking, the joint distribution is $p(y, z|x)$ and the two distributions of interest are the observations $p(y|z)$ and encoding $p(z|x)$. This high-level decomposition will hold for the entirety of the thesis. Each chapter in the thesis will explore a different low-level decomposition of the observation or encoding distributions in addition to how those distributions parameterized. We give three examples of low-level decompositions below.

Example 2.4.4. Autoregressive models (AR): A common modeling assumption is that the output y is modeled autoregressively. Autoregressive models use the chain rule to factor

$$p(y) = \prod_t p(y_t \mid y_{<t})$$

over time. In a state tracking model, this would be

$$p(y, z \mid x) = p(y \mid z)p(z \mid x) = \prod_t p(y_t \mid y_{<t}, z)p(z \mid x).$$

This makes no conditional independence assumptions other than $y \perp x \mid z$, allowing models to express complicated relationships in the output y and state z. We will use this model in chapter 5. //

Example 2.4.5. Naive Bayes (NB): A simpler model is naive Bayes, which assumes that both the output y and all inputs $x = (x_1, \ldots, x_T)$ are generated conditionally independently given the latent state z. The joint distribution is given by

$$p(x, y, z) = p(z)p(y \mid z) \prod_t p(x_t \mid z).$$

We will use this model in chapters 6 and 7. //

Example 2.4.6. Hidden Markov Models (HMM): An example of a model with structured constraints is the hidden Markov model, which we will explore in chapters 3 and 4. HMMs follow a generative process where a sequence of states are first generated, then each state in that sequence emits an output conditionally independently of all other states and output. The HMM makes two strong assumptions when predicting the next word y given all previous words $x = (x_1, \ldots, x_T)$: (1) The next word y depends only on the state representation z_T and (2) that there is a sequential state bottleneck: the information from x must transition from the previous state representation z_{t-1} to z_t. This creates a bottleneck for information flow from x to y. The factorization is given by

$$p(y, z, x) = p(y \mid z_T) p(z_T \mid z_{T-1}) \prod_t p(x_t|z_t) p(z_t \mid z_{t-1}).$$

Note that the joint distribution of this model also includes the input tokens x. We will use this model in chapters 3 and 4. //

2.5 Inference

The goal of probabilistic inference is to use the probabilistic model to answer questions or queries. Queries are quantities and distributions derived from the joint distribution. For example: Given the past words x and the observed next word y, what should the state representation z have been? Had not observed y, what would the state be? Such questions are central to learning and reasoning. In learning, we must optimize a model to fit the inputs and outputs; however, the connection between the inputs and outputs are mediated by the unobserved state. We must therefore marginalize over the possible latent state values via marginal inference. For reasoning, we must update our beliefs over the latent state given new observed outputs. This is accomplished via conditional inference. This section will introduce the machinery necessary for learning and reasoning.

2.5.1 Learning and marginal inference

The goal of learning is to find the distribution $p(y|x)$ that best explains the outputs y given the inputs x. In the case of latent state models, this requires performing marginal inference to propagate the uncertainty over the unobserved state z into the outputs y.

Formally, prediction in discrete latent variable models requires computing the marginal distribution

$$p(y|x) = \sum_z p(y|x,z)p(z|x).$$

We can also express the marginal distribution in matrix notation. Let $O_{i,j} = p(y = i|x, z = j)$ and $g_j = p(z = j|x)$. Computing this term in general can be challenging if there are many possible values of z, as we must evaluate the likelihood $p(y|z)$ for each value of z. In this thesis, we primarily consider settings where z can be marginalized exactly or approximated efficiently.

Given the marginal distribution $p(y|x)$, we perform maximum likelihood estimation to find the distribution that best fits the data. The goal is to maximize the likelihood with respect to the parameters: $\text{argmax}_\theta \log p(y|x;\theta)$. We achieve this via minibatch stochastic gradient ascent, where we estimate the gradient

$$\begin{aligned}
\nabla_\theta \log p(y|x;\theta) &= \frac{1}{p(y|x;\theta)} \nabla p(y \mid x;\theta) \\
&= \frac{1}{p(y|x;\theta)} \nabla_\theta \sum_z p(y, z \mid x;\theta) \\
&= \frac{1}{p(y|x;\theta)} \sum_z p(y, z \mid x;\theta) \nabla_\theta \log p(y, z \mid x;\theta) \\
&= \sum_z p(z \mid x, y;\theta) \nabla_\theta \log p(y, z \mid x;\theta) \\
&= \mathbb{E}_{p(z|x,y;\theta)} \left[\nabla_\theta \log p(y \mid x, z;\theta) + \nabla_\theta \log p(z \mid x;\theta) \right].
\end{aligned}$$

In the case of exact inference, the gradient estimator can be computed efficiently using automatic differentiation. We will also resort to simple approximations of the marginal likelihood gradient estimator: In chapter 5, we will replace the posterior expectation with a simple top-k operation.

2.5.2 Updating uncertainty through conditional inference

While marginal inference is concerned with propagating uncertainty through the latent state, the goal of conditional inference is to update uncertainty over the state by utilizing information from the output to inform the latent state. Formally, given a prior state distribution $p(z|x)$, we can update our uncertainty after observing y via

$$p(z \mid x, y) = \frac{p(y, z \mid x)}{p(y \mid x)}.$$

Example 2.5.1. Consider the naive Bayes' model (Example 2.4.5). The posterior simplifies to

$$p(z \mid x, y) = \frac{p(y, z, x)}{p(y, x)} = \frac{p(y|z) \prod_t p(x_t \mid z) p(z)}{\sum_z p(y|z) \prod_t p(x_t \mid z) p(z)}.$$

//

2.6 Parameterization

In this section, we introduce how to parameterize conditional probability distributions. The conditional distribution specifies the probability of outputs given inputs, $p(y|x;\theta)$, parameterized by a vector, matrix, or tensor θ. We break from the notation of the previous sections and only consider inputs x and outputs y – in the earlier sections, as well as future sections, we can treat the state as an input or output in the corresponding conditional distribution.

We consider two types of parameterizations: scalar and neural. Scalar parameterizations of $p(y|x;\theta)$ associate one scalar value with every input and output combination $\theta_{x,y} \in \mathbb{R}$, resulting in a matrix $\theta \in \mathbb{R}^{\mathcal{X} \times \mathcal{Y}}$. Concretely, this can be implemented as

$$p(y \mid x; \theta) = \frac{\exp(\theta_{x,y})}{\sum_{y'} \exp(\theta_{x,y'})} = \text{softmax}(\theta_x)_y,$$

where the softmax function normalizes the values of θ_x to sum to 1. In the case of language modeling, the space of input sequences $x \in \mathcal{X}^*$ is infinite resulting in an infinite dimensional model. Infinite dimensional models, as well as models with extremely high but finite dimension, are cumbersome for two reasons: First, representing and storing the parameters is costly. Second, larger models often require larger datasets to fit efficiently (Hoffmann et al., 2022). In the remainder of this section, we describe neural parameterizations, which generate the scalar parameters dynamically for specific inputs x.

Neural networks can dynamically generate high-dimensional probability distributions from low-dimensional weight spaces. The parameters of the distribution $p(y|x;\theta)$ are generated by a neural network f with weights w if each element of

$$\theta_{x,y} = f(x, y, w),$$

where $\dim(w) < \dim(\theta)$.

Neural networks are composed of a small set of building blocks. The main building blocks necessary for this thesis are embeddings, feedforward networks, normalization, and attention. We use these building blocks to map language, represented as a sequence of tokens x, to the parameters of a conditional distribution θ. The primary challenge in representing language is dimensionality reduction. This is caused by two factors: Sequences in language vary in length and each token in the sequence is drawn from a high-dimensional, discrete set. We start with neural parameterizations of conditional distributions that take a single input, $p(y|x)$, where x is a singleton element. Then, we generalize that to settings where x is a sequence of variable length.

2.6.1 Singleton input

We start by introducing a neural parameterization of $p(y|x;\theta)$ with singleton input $x \in \mathcal{X}$, output $y \in \mathcal{Y}$, and parameters $\theta \in \mathbb{R}^{\mathcal{X}\times\mathcal{Y}}$. The neural parameterization will use a neural network to generate the parameters of the distribution $p(y|x;\theta)$ with $O(|\mathcal{X}| + |\mathcal{Y}|)$ weights rather than $O(|\mathcal{X}| \cdot |\mathcal{Y}|)$ parameters.

A simple neural parameterization, which forms the basis of chapters 3 and 4, first embeds the input, applies a multi-layer perceptron (MLP), takes the inner-product with output embeddings, then applies the softmax function:

$$\theta_{x,y} = \mathrm{softmax}(\langle \mathrm{MLP}(\mathrm{emb}(x)), \mathrm{emb}(y) \rangle).$$

Embeddings reduce the dimensionality of the input by embedding $x \in \mathcal{X}$ to a vector $e \in \mathbb{R}^d$. This is achieved by associating each token type with a learnable embedding vector: $\mathrm{emb}(x) \in \mathbb{R}^d$. This can be implemented by storing a matrix of all embeddings $E = \left(\mathrm{emb}\left(x_1\right), \ldots, \mathrm{emb}\left(x_{|\mathcal{X}|}\right)\right) \in \mathbb{R}^{|\mathcal{X}| \times d}$, and indexing to retrieve embeddings $E_x = \mathrm{emb}(x)$. The output embeddings are symmetric: $E_y = \mathrm{emb}(y) \in \mathbb{R}^d$.

MLPs serve to learn expressive, nonlinear relationships and are given by

$$\mathrm{MLP}(\mathrm{emb}(e)) = \sigma(W\mathrm{emb}(x) + b),$$

which applies a nonlinearity such as $\sigma(\cdot) = \max(\cdot, 0)$, known as the Rectified Linear Unit (ReLU). The weights $W, b \in \mathbb{R}^{d \times d} \times \mathbb{R}^d$ are learnable. Transforming the input embedding with an MLP, then taking the inner-product with the output embedding serves to learn an affinity measure between the input and output values x and y.

Finally, we remind the reader of the definition of softmax, which normalizes the affinity measures to a probability distribution:

$$\mathrm{softmax}(x) = \frac{\exp(x)}{\sum_{x'} \exp(x')}.$$

The learnable weights of the network can be optimized via automatic differentiation with respect to the parameters of the distribution θ.

2.6.2 Sequential input: $x \in \mathcal{X}^*$

Our goal in this section is to generate the parameters of probability distributions, $p(y|x;\theta)$, that condition on sequences, $x = (x_1, \ldots, x_T) \in \mathcal{X}^T$, with a neural network: $\theta = f(x, y)$. There are two challenges with sequential inputs: First, that we cannot generate the full parameters $\theta \in \mathbb{R}^{\mathcal{X}^* \times \mathcal{Y}}$ as there are infinitely many possible sequences x. Second, we must use a neural network that can be applied to variable-length sequences. We solve the first challenge by generating the parameters for a single row $\theta_x = f(x, \cdot)$, and the second by introducing a pooling operation called attention and combining it with the parameterization in the previous section. This neural parameterization is a simplification of the models used in chapter 2.4.4, as well as modern neural methods for natural language processing (Bahdanau et al., 2015, Vaswani et al., 2017). For details on the full model, known as a Transformer, please refer to Vaswani et al. (2017).

Similar to the previous case of singleton inputs, we start by embedding the inputs $x_t \in \mathcal{X}$ independently, yielding a sequence of vectors $e \in \mathbb{R}^{T \times d}$. Concatenating all token embeddings into a matrix $E \in \mathbb{R}^{|\mathcal{V}| \times d}$, we can lift the embedding operation to the sequence level:

$$E[x] = [\mathrm{emb}(x_1), \ldots, \mathrm{emb}(x_T)].$$

Given the sequence embeddings $E[x]$, we must pool their representations to obtain a single representation h from which to make a prediction. We use the attention operator, introduced in Bahdanau et al. (2015) and popularized by Vaswani et al. (2017),

$$\mathrm{attn}(q, K, V) = \mathrm{softmax}\left(\frac{qK^\top}{\sum qK^\top}\right) V.$$

We can then produce a distribution over outputs by applying the attention operator to transformations of the embedded input:

$$p(y|x) = \text{softmax}(E[y]^\top \text{MLP}(\text{attn}(W_q E[x], W_k E[x], W_v E[x]))),$$

where $W_q, W_k, W_v \in \mathbb{R}^{d \times d}$ are learnable weight matrices. Intuitively, the attention operator produces an input-dependent, or query-dependent, average of the inputs, pooling the information from a variable-length sequence into a single input-aware vector representation of the sequence.

2.7 Large language models and prompting

Large language models (LLMs) are autoregressive models of text, trained on large amounts of data. While not the focus of this thesis, it is worth noting that in recent years LLMs have become increasingly important due to their ability to capture domain knowledge from large text corpora.

In Chapter 7, we rely on LLMs as powerful conditional distributions capable of tasks such as instruction following and semantic parsing without any further training. The best-performing LLMs are either too large to train or fine-tune on standard hardware or are only accessible through a limited API. Instead, we rely on prompting.

As autoregressive models, LLMs generate tokens from left to right. Prompting takes advantage of this by giving LLMs an instruction and/or set of examples as input prefix x, then the LLM generates the output completion y. There are two styles of prompting: Zeroshot prompting, where the LLM is given a natural language instruction x, such as $x =$ 'Please complete this sentence: The cat in the', and fewshot prompting, where the LLM is given a set of examples in x and possibly instructions as well.

2.8 Thesis roadmap

This thesis explores different state tracking representations for three applications: language modeling, question answering, and task-oriented dialogue. Table 2.1 outlines the application and representation covered in each chapter.

Ch.	Title	Application	Representation
3	Scaling Hidden Markov Language Models	LM	Integer
4	Low-Rank Constraints For Fast Inference in Structured Models	LM	Integer
5	HOP, UNION, GENERATE: Explainable Multi-hop Reasoning without Rationale Supervision	QA	Text
6	Modeling Perspective-Dependent Ambiguity in Collaborative Dialogue	Dialogue	Symbols
7	Symbolic Planning and Code Generation for Grounded dialogue	Dialogue	Symbols and code

Table 2.1: Summary of the applications and state tracking representations explored in each chapter.

3

Scaling Hidden Markov Language Models

3.1 Introduction

We begin our exploration of state tracking representations with language modeling. This chapter proposes a discrete, compositional state tracking representation for language modeling known as the hidden Markov model (Example 2.4.6). Hidden Markov models (HMMs) are a fundamental latent-variable model for sequential data, with a rich history in NLP. They have been used extensively in tasks such as tagging (Merialdo, 1994), alignment (Vogel et al., 1996), and language modeling (Kuhn et al., 1994, Huang, 2011). While once popular due to their explicit uncertainty modeling and computational tractability, HMMs have been replaced by more complex sequence models have been introduced such as neural language models (Zaremba et al., 2014, **?**).

Our goal in this chapter is to revisit HMMs, scaling them to unprecedented size. We improve the performance of HMMs by taking several lessons from the success of neural models for NLP tasks: (a) model size is critical for accuracy, e.g. large LSTMs Zaremba et al. (2014) show marked improvements in performance; (b) the right parameterization is critically important for representation learning, e.g. a feedforward model Bengio et al. (2003) can have the same distributional assumptions as an n-gram model while performing significantly better; (c) dropout is key to achieving strong performance (Zaremba et al., 2014, Merity et al., 2017). We materialize these lessons by introducing three techniques: a sparse modeling constraint that allows us to use a large number of hidden states while maintaining efficient exact inference, a neural parameterization that improves generalization while remaining faithful to the probabilistic structure of the HMM, and a variant of dropout that both improves accuracy and halves the computational overhead during training.

The resulting approach allows us to train an HMM with tens of thousands of states while maintaining efficiency and significantly outperforming past HMMs as well as classical n-gram language models.

3.2 Background: HMMs

We are interested in learning a distribution over observed tokens $\mathbf{x} = \langle x_1, \ldots, x_T \rangle$, with each token x_t an element of the finite vocabulary $\mathcal{X}$. Hidden Markov models (HMMs) specify a joint distribution over observed tokens $\mathbf{x}$ and discrete latent states

$\mathbf{z} = \langle z_1, \ldots, z_T \rangle$, with each z_t from the finite set $\mathcal{Z}$. For notational convenience, we define the starting state $z_0 = \epsilon$. This yields the joint distribution

$$p(\mathbf{x}, \mathbf{z}; \theta) = \prod_{t=1}^{T} p(x_t \mid z_t) p(z_t \mid z_{t-1}). \tag{3.1}$$

We refer to the transition and emission matrices as the distributional parameters of the HMM. Specifically, let $\mathbf{A} \in [0,1]^{|\mathcal{Z}| \times |\mathcal{Z}|}$ be the transition probabilities and $\mathbf{O} \in [0,1]^{|\mathcal{Z}| \times |\mathcal{X}|}$ the emission probabilities

$$p(z_t \mid z_{t-1}) = A_{z_{t-1} z_t} \quad p(x_t \mid z_t) = O_{z_t x_t}. \tag{3.2}$$

We distinguish between two types of model parameterizations: *scalar* and *neural*, where the model parameters are given by θ. A scalar parameterization sets the model parameters equal to the distributional parameters, so that $\theta = \{\mathbf{A}, \mathbf{O}\}$, resulting in $O(|\mathcal{Z}|^2 + |\mathcal{Z}||\mathcal{X}|)$ model parameters. A neural parameterization instead generates the distributional parameters from a neural network (with parameters θ), decoupling the size of θ from $\mathbf{A}, \mathbf{O}$. This decoupling gives us the ability to choose between compact or overparameterized θ (relative to $\mathbf{A}, \mathbf{O}$). As we scale to large state spaces, we take advantage of compact neural parameterizations.

In order to fit an HMM to data $\mathbf{x}$, we must marginalize over the latent states to obtain the likelihood $p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z})$. This sum can be computed in time $O(T|\mathcal{Z}|^2)$ via the forward algorithm, which becomes prohibitive if the number of latent states $|\mathcal{Z}|$ is large. We can then optimize the likelihood with gradient ascent (or alternative variants of expectation maximization).

HMMs and RNNs Although the forward algorithm resembles that of the forward pass in a recurrent neural network (RNN) (Buys et al., 2018), there are key representational differences. RNNs do not decouple the latent dynamics from the observed. This often leads to improved accuracy, but precludes posterior inference which is useful for interpretability. A further benefit of HMMs over RNNs is that their associative structure allows for parallel inference via the prefix-sum algorithm Ladner & Fischer (1980).[1] Finally, HMMs bottleneck information from every timestep through a discrete hidden state. NLP has a long history of utilizing discrete representations, and discrete representations may yield interesting results. For example, recent work has found that discrete latent variables work well in low-resource regimes (Jin et al., 2020).

3.3 Scaling HMMs

We propose three extensions to scale HMMs for better language modeling performance: blocked emissions, which allow for very large models; neural parameterization, which makes it easy for states to share model parameters; and state dropout, which encourages broader state usage.

Blocked Emissions Our main goal is to apply a HMM with a large number of hidden states to learn the underlying dynamics of language data. However, the $O(T|\mathcal{Z}|^2)$ complexity of marginal inference practically limits the number of HMM states. We can get around this limit by making an assumption on the HMM emission matrix $\mathbf{O}$. As noted by Dedieu et al. (2019), restricting the number of states that can produce each word can improve inference complexity. We utilize a slightly stronger assump-

[1]Quasi-RNNs (Bradbury et al., 2016) also have a (parallel) logarithmic dependency on T by applying the same prefix-sum trick, but do not model uncertainty over latent dynamics.

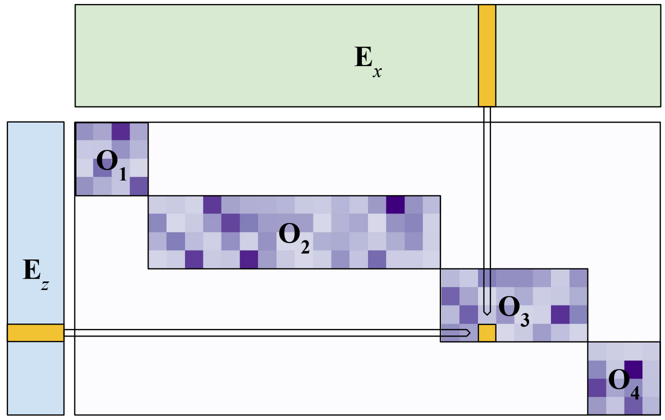


Figure 3.1: The emission matrix as a set of blocks $\mathbf{O}_1, \dots, \mathbf{O}_4$ with fixed number of states k. The columns of each block may vary, as there is no constraint on the number of words a state can emit. Each non-zero cell is constructed from an MLP applied to word $\mathbf{E}_x$ and state $\mathbf{E}_z$ embeddings.

tion on the model: a) states are partitioned into M equal sized groups each of which emit the same subset of words, and b) each word is only admitted by one group of $k = |\mathcal{Z}|/M$ states which we indicate as $\mathcal{Z}_x \subset \mathcal{Z}$.

We implement this group structure through a set of blocked emissions, each corresponding to one of the M state groups,

$$\mathbf{O} = \begin{bmatrix} \mathbf{O}_1 & 0 & 0 \\ 0 & \dots & 0 \\ 0 & 0 & \mathbf{O}_M \end{bmatrix},$$

where $\mathbf{O}_m \in \mathbb{R}^{k \times |\mathcal{X}_m|}$. Figure 3.1 shows these emission blocks. Each block matrix $\mathbf{O}_m$ gives the probabilities for emitting tokens $\mathcal{X}_m$ for states in group m, i.e. states $(m-1)k$ through mk.

With this constraint, exact marginalization can be computed via

$$p(\mathbf{x}) = \sum_{z_1 \in \mathcal{Z}_{x_1}} p(z_1 \mid z_0) p(x_1 \mid z_1) \times \cdots \sum_{z_T \in \mathcal{Z}_{x_T}} p(z_T \mid z_{T-1}) p(x_T \mid z_T) \tag{3.3}$$

Since there are only k states with nonzero probability of occurring at every timestep, we only need to consider transitioning from the $|\mathcal{Z}_{x_t}| = k$ previous states to the next $|\mathcal{Z}_{x_{t+1}}| = k$ states, resulting in $O(k^2)$ operations per timestep. This gives a serial complexity of $O(Tk^2)$.[2]

Neural Parameterization A larger state space allows for longer HMM memory, but also requires more parameters. Even with blocked emissions, the scalar model parameterization of an HMM grows as $O(|\mathcal{Z}|^2)$ due to the transition matrix. A neural parameterization allows us to share parameters between words and states to capture common structure.

Our parameterization uses an embedding for each state in $\mathcal{Z}$ ($\mathbf{E}_z \in \mathbb{R}^{|\mathcal{Z}| \times h}$) and each token in $\mathcal{X}$ ($\mathbf{E}_x \in \mathbb{R}^{|\mathcal{X}| \times h}$). From these we can create representations for leaving and entering a state, as well as emitting a word:

$$\mathbf{H}_{\text{out}}, \mathbf{H}_{\text{in}}, \mathbf{H}_{\text{emit}} = \text{MLP}(\mathbf{E}_z)$$

[2]This can be sped up on a parallel machine to $O(\log(T)k^2)$ via a binary reduction.

Algorithm 1 HMM Training (a single batch)

Given: block structure and model parameters
Sample block-wise dropout mask $\mathbf{b}$
Compute $\mathbf{A}, \mathbf{O}$ ignoring $b_z = 0$
for all examples $\mathbf{x}$ in batch **do**
    Compute $\log p(\mathbf{x}; \mathbf{A}, \mathbf{O})$
    Compute grad wrt parameters of $\log p(\mathbf{x})$
Update model parameters $\mathbf{E_z}, \mathbf{E_x}$ and MLP

with all in $\mathbb{R}^{|\mathcal{Z}|\times h}$. The HMM distributional parameters are then computed as[3]

$$\mathbf{O} \propto \exp\left(\mathbf{H}_{\text{emit}}\mathbf{E}_x^\top\right) \qquad \mathbf{A} \propto \exp\left(\mathbf{H}_{\text{in}}\mathbf{H}_{\text{out}}^\top\right). \tag{3.4}$$

The MLP architecture follows Kim et al. (2019), with details in the appendix. This factorized parameterization, shown in Figure 3.1, reduces the total parameters to $O(h^2 + h|\mathcal{Z}| + h|\mathcal{X}|)$.

Note that parameter computation is independent of inference and can be cached completely as the emission and transition matrices, $\mathbf{A}$ and $\mathbf{O}$, at test-time. For the training algorithm, shown in Algorithm 1, we compute $\mathbf{A}$ and $\mathbf{O}$ once per batch while RNNs and similar models recompute emissions every token.

Dropout as State Reduction Finally, to encourage full use of the large state space, we introduce dropout that prevents the model from favoring specific states. We propose a form of HMM state dropout that removes states from use entirely at each batch, which also has the added benefit of speeding up inference.

State dropout acts on each emission block $\mathbf{O}_1, \ldots, \mathbf{O}_M$ independently. For each

[3] As an optimization, one could only compute the nonzero emission matrix blocks saving space and time. In practice we compute the full matrix as in the equation.

block, we sample a binary dropout mask by sampling λk dropped row indices uniformly without replacement, where λ is the dropout rate. We concatenate these into a global vector $\mathbf{b} \in \{0,1\}^{|\mathcal{Z}|}$, which, along with the previous constraints, ensures,

$$\begin{aligned} p(z_t \mid z_{t-1}) &\propto b_{z_t} A_{z_{t-1} z_t} \\ p(x_t \mid z_t) &\propto b_{z_t} 1(z \in \mathcal{Z}_{x_t}) O_{z_t x_t} \end{aligned} \tag{3.5}$$

An example of the HMM lattice after state dropout is show in Figure 3.2.

In addition to accuracy improvements, state dropout gives a large practical speed up for both parameter computation and inference. For $\lambda = 0.5$ we get a $4\times$ speed improvement for both, due to the reduction in possible transitions. This structured dropout is also easy to exploit on GPU, as it maintains block structure.

3.4 Experimental Setup

Emission Blocks The model requires partitioning token types into blocks $\mathcal{X}_m$. While there are many partitioning methods, a natural choice is Brown clusters (Brown et al., 1992, Liang, 2005) which are also based on HMMs. Brown clusters are obtained by assigning every token type in $\mathcal{X}$ a state in an HMM, then merging states until a desired number of partitions M is reached. We construct the Brown clusters on the training portions of the datasets and assume the vocabulary remains identical at test time (with OOV words mapped to unk). We include more background on Brown Clusters in the appendix.

State Dropout We use a dropout rate of $\lambda = 0.5$ at training time. For each block of size $|\mathcal{X}_m|$, we sample $\lambda|\mathcal{X}_m|$ states to use in that block each batch. We draw states from

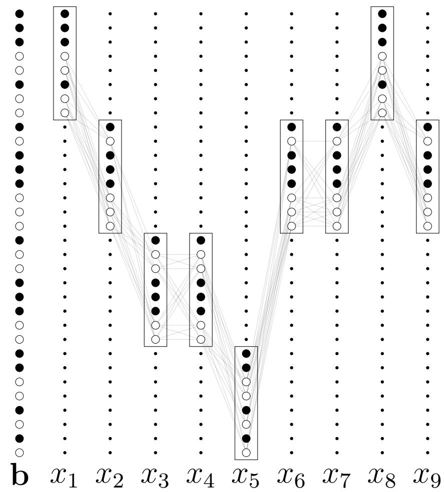


Figure 3.2: The computation of $p(\mathbf{x})$ is greatly reduced by blocked emissions and state dropout. In the above trellis, each row corresponds to a latent state and each column after the first to a timestep. Each edge between nodes corresponds to a nonzero transition probability. Blocked emissions result in a small subset of all states emitting a given word, as shown by the rectangles. State dropout (leftmost column) allows us to further reduce the number of states we consider, halving the number of (white) states that have nonzero probability in each rectangle. In experiments, the number of possible transitions may be as large as 2^{30} while the max number of non-zero transitions is 2^{16}.

each block from a multivariate hypergeometric distribution using the Gumbel Top-k trick for sampling without replacement (Vieira, 2014). At test time we do not use state dropout.

Datasets We evaluate on the Penn Treebank (Marcus et al., 1993) (929k train tokens, 10k vocab) and Wikitext2 (Merity et al., 2016) (2M train tokens, 33k vocab) datasets. For Penn Treebank we use the preprocessing from Mikolov et al. (2011), which lowercases all words and substitutes OOV words with unks. We insert EOS tokens after each sentence. For Wikitext2 casing is preserved, and all OOV words are unked. We insert EOS tokens after each paragraph. In both datasets OOV words were included in the perplexity (as unks), and EOS was included in the perplexity as well

(Merity et al., 2017).

Baselines Baselines include both state-of-the-art language models and other alternative LM styles. These include AWD-LSTM (Merity et al., 2017); a 900-state scalar HMM and HMM+RNN extension, which discards the HMM assumptions (Buys et al., 2018); a traditional Kneser-Ney 5-gram model (Mikolov & Zweig, 2012, Heafield et al., 2013), a 256 dimension feedforward neural model, and a 2-layer 256 dimension LSTM.

We compare these with our approach: the very large neural HMM (VL-HMM). Unless otherwise noted, our model has $|\mathcal{Z}| = 2^{15}$ total states but only considers $k = 256$ states at every timestep at test time with $M = 128$ groups.[4] The state and word embeddings as well as the MLP have a hidden dimension of 256. We train with a state dropout rate of $\lambda = 0.5$. See the appendix for all hyperparameters.

3.5 Results

Table 3.1 gives the main results. On PTB, the VL-HMM is able to achieve 125.0 perplexity on the valid set, outperforming a FF baseline (159.9) and vastly outperforming the 900-state HMM from Buys et al. (2018) (284.6).[5] The VL-HMM also outperforms the HMM+RNN extension of Buys et al. (2018) (142.3). These results indicate that HMMs are a much stronger model on this benchmark than previously claimed. However, the VL-HMM is still outperformed by LSTMs which have been extensively studied for this task. This trend persists in Wikitext-2, with the VL-HMM outper-

[4]The 256 dim FF, LSTM, and VL-HMM in particular have comparable computational complexity: $O(256^2 T)$.

[5]Buys et al. (2018) only report validation perplexity for the HMM and HMM+RNN models, so we compare accordingly.

Model	Param	Val	Test
PENN TREEBANK			
KN 5-gram	2M	-	141.2
AWD-LSTM	24M	60.0	57.3
256 FF 5-gram	2.9M	159.9	152.0
2x256 dim LSTM	3.6M	93.6	88.8
HMM+RNN	10M	142.3	-
HMM $\|\mathcal{Z}\| = 900$	10M	284.6	-
VL-HMM $\|\mathcal{Z}\| = 2^{15}$	11.4M	125.0	116.0
WIKITEXT			
KN 5-gram	5.7M	248.7	234.3
AWD-LSTM	33M	68.6	65.8
256 FF 5-gram	8.8M	210.9	195.0
2x256 LSTM	9.6M	124.5	117.5
VL-HMM $\|\mathcal{Z}\| = 2^{15}$	17.3M	166.6	158.2

Table 3.1: Perplexities on PTB / WIKITEXT-2. The HMM+RNN and HMM of Buys et al. (2018) reported validation perplexity only for PTB.

forming the FF model but underperforming an LSTM.

Figure 3.3 examines the effect of state size: We find that performance continuously improves significantly as we grow to 2^{16} states, justifying the large state space. The marginal improvement does lower as the number of states increases, implying that the current approach may have limitations in scaling to even larger state spaces.

Table 3.2 considers other ablations: Although neural and scalar parameterizations reach similar training perplexity, the neural model generalizes better on validation with almost 100x fewer model parameters. We find that state dropout results in both an improvement in perplexity and a large improvement in computational speed. See the appendix for emission sparsity constraint ablations, as well as experiments on further reducing the number of parameters.

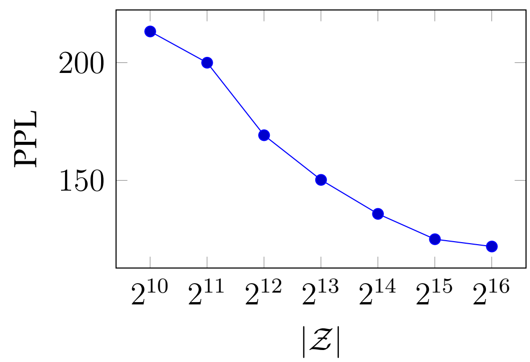


Figure 3.3: Perplexity on PTB by state size $|\mathcal{Z}|$ ($\lambda = 0.5$ and $M = 128$).

Model	Param	Train	Val	Time
VL-HMM (2^{14})	7.2M	115	134	40
- neural param	423M	119	169	14
- state dropout	7.2M	88	157	100

Table 3.2: Ablations on PTB ($\lambda = 0.5$ and $M = 128$) with a smaller model $|\mathcal{Z}| = 2^{14}$. Time is ms per eval batch (Run on RTX 2080). Ablations were performed independently, removing a single component per row. Removing the neural parameterization results in a scalar parameterization.

3.6 Related Work

In order to improve the performance of HMMs on language modeling, several papers have combined HMMs with neural networks. Buys et al. (2018) develop an approach to relax HMMs, but their models either perform poorly or alter the probabilistic structure to resemble an RNN. Krakovna & Doshi-Velez (2016) utilize model combination with an RNN to connect both approaches in a small state-space model. Our method instead focuses on scaling pure HMMs to a large number of states.

Prior work has also considered neural parameterizations of HMMs. Tran et al. (2016) demonstrate improvements in POS induction with a neural parameterization of an HMM. They consider small state spaces, as the goal is tag induction rather than language modeling.[6]

Most similar to this work are the large HMM models of Dedieu et al. (2019). They introduce a sparsity constraint in order to train a 30K state non-neural HMM for character-level language modeling; however, their constraint precludes application to large vocabularies. We overcome this limitation and train models with neural parameterizations on word-level language modeling.

Finally, another approach for scaling state spaces is to grow from small to big via a split-merge process (Petrov et al., 2006, Huang, 2011). In particular, Huang (2011) learn an HMM for language modeling via this process. As fixed-size state spaces are amenable to batching on modern hardware, we leave split-merge procedures for future work.

3.7 Conclusion

Our method demonstrates that we can effectively scaling HMMs to large state spaces on parallel hardware, and results in accuracy gains compared to other HMM models. In order to scale, we introduce three techniques: a blocked emission constraint, a neural parameterization, and state dropout, which lead to an HMM that outperforms n-gram models and prior HMMs. Once scaled up to take advantage of modern hardware, very large HMMs demonstrate meaningful improvements over smaller HMMs. HMMs are a useful class of probabilistic models with different inductive biases, performance characteristics, and conditional independence structure than RNNs.

The success of scaling HMMs for language modeling motivates the next chapter,

[6]Other work has used neural parameterization for structured models, such as dependency models (Han et al., 2017), hidden semi-Markov models (Wiseman et al., 2018), and context free grammars (Kim et al., 2019).

which introduces a more general constraint that applies to a wider class of state tracking models for language modeling.

4

Low-Rank Constraints for Fast Inference in Structured Models

4.1 Introduction

The second stop in our exploration of state tracking representations extends the previous section's exploration of language modeling representations to a more general class of representations. The emission sparsity assumption of the previous chapter is limited to shallow models such as hidden Markov models (HMMs) and does not extend to hierarchical latent representations such as trees. In this chapter, we propose a low-rank constraint that applies to a wide class of language models with intepretable state tracking representations. The constraint applies not only to HMMs, but also extends to hierarchical state tracking representations afforded by hidden semi-Markov models (HSMM) and probabilistic context-free grammars (PCFG).

Hierarchical state tracking representations are more expressive than shallow ones, but the increased expressivity comes at a cost. Inference with hierarchical representations is computationally expensive – while inference in HMMs is quadratic, inference in PCFGs is cubic which limits the ability to reach a massive scale. Promisingly, recent work has shown that in specific situations these models can be scaled, and that the increased scale results in commensurate improvements in accuracy – without sacrificing the ability to perform exact inference (Dedieu et al., 2019, Chiu & Rush, 2020b, Yang et al., 2021b).

In this chapter, we propose an approach for improving the runtime of a large class of structured latent models by introducing a low-rank constraint. We target the family of models where inference can be formulated through a labeled directed hypergraph, which describes a broad class of dynamic-programming based inference (Klein & Manning, 2004, Huang & Chiang, 2005, Zhou et al., 2006, Javidian et al., 2020, Chiang & Riley, 2020). We show how under low-rank constraints these models allow for more efficient inference. Imposing a low-rank constraint allows for a key step of inference to be rewritten as a fast matrix-vector product. This approach is also inspired by recent advances in computationally efficient neural attention attention (Katharopoulos et al., 2020, Peng et al., 2021, Choromanski et al., 2020), a significantly different task and formulation, that rewrites matrix-vector products as fast low-rank products using approximate kernel techniques.

We evaluate this approach by learning low-rank structured models for the tasks of language modeling, polyphonic music modeling, unsupervised grammar induction, and video modeling. For these tasks we use a variety of models including HMMs, PCFGs, and HSMMs. As the application of low-rank constraints is nontrivial in high-dimensional structured models due to reduced expressivity, we demonstrate effective techniques for overcoming several practical challenges of low-rank parameterizations. We find that our approach achieves very similar results to unconstrained models at large state sizes, while the decomposition allows us to greatly increase the speed of inference. Results on HMMs show that we can scale to more than 16,000 states; results on PCFGs achieve a significant perplexity reduction from much larger state spaces compared to past work (Kim et al., 2019); and results on HSMMs show that our formulation enables scaling to much larger state spaces for continuous emissions (Fried et al., 2020).

4.2 Background: Latent Structure and Hypergraphs

We consider the problem of modeling a sequence of observations $p(x) = p(x_1, \ldots, x_T)$. These observations can range in complexity from the words in a sentence to a series of co-occurring musical notes, or to features of video frames, and may be discrete or continuous. We assume these observations are generated by an unobserved (latent) structured representation z, and therefore model the joint $p(x, z)$. The structure may be sequential or hierarchical, such as latent trees, and the set of structures $\mathcal{Z}$ is combinatorial, i.e. exponential in size with respect to the input sentence. In order to train these models on observations, we must optimize the evidence $p(x) = \sum_z p(x, z)$ by marginalizing over z. Scaling this marginalization is the focus of this work.

Hypergraphs are a graphical model formalism for structured distributions that admit tractable inference through dynamic programming (Klein & Manning, 2004, Huang & Chiang, 2005, Zhou et al., 2006, Javidian et al., 2020, Chiang & Riley, 2020).[1] A labeled, directed, acyclic hypergraph consists of a set of nodes $\mathcal{V}$, a set of hyperedges $\mathcal{E}$, and a designated root node $S \in \mathcal{V}$. Each node $v \in \mathcal{V}$ has a collection of labels $\mathcal{L}_v$. Each hyperedge $e \in \mathcal{E}$ has a head node u and tuple of tail nodes, $v = (v_1, \ldots, v_{|e|})$, where $|e|$ is the number of tail nodes. For simplicity, we will assume *at most* 2 tail nodes v_1, v_2, and unless noted, a fixed label set $\mathcal{L}$ throughout. Each hyperedge e is associated with a score matrix $\Psi_e \in \mathbb{R}^{\mathcal{L} \times \mathcal{L}^{|e|}}$ with a score for all head and tail labels.[2] We use the notation $[\Psi_e]_{z_u,(z_1,z_2)}$ to indicate the score for head label z_u and tail labels z_1 and z_2. Finally, we assume we have a topological ordering over the edges.

A hypergraph is used to aggregate scores bottom-up through a dynamic programming (belief propagation) algorithm. Algorithm 2 shows the algorithm. It works by filling in a table vector $\alpha_v \in \mathbb{R}^{\mathcal{L}}$ for each node v in order, and is initialized to 1 at the leaf nodes.[3] It returns the sum over latent structures, $p(x)$. Counting loops, the worst-case runtime complexity is $O(|\mathcal{E}| \times L^{|e^*|+1})$ where $L = |\mathcal{L}|$ is the size of the label set and $|e^*|$ the max hyperedge tail size. Algorithm 3 shows the same algorithm in matrix form by introducing joined tail vectors $\beta_v \in \mathbb{R}^{\mathcal{L}^{|e|}}$ for each group of nodes v. Letting $z_v = (z_1, z_2)$, the joined tail vector contains entries $[\beta]_{z_v} = [\alpha_{v_1}]_{z_1}[\alpha_{v_2}]_{z_2}$.

To make this formalism more concrete, we show how hypergraphs can be used for inference in several structured generative models: hidden Markov models, probabilistic context-free grammars, and hidden semi-Markov models. Inference in these

[1] While the formalism is similar to undirected factor graphs, it allows us to represent more complex distributions: notably dependency structures with unknown topologies, such as latent trees.

[2] This formalism can represent inference in both locally and globally normalized models, although we focus on local normalization in this work.

[3] The accumulation of scores is denoted by $\overset{+}{\leftarrow}$. Multiple hyperedges can have the same head node, whose scores must be added together.

Algorithm 2 Hypergraph marginalization (Scalar form)

for $u \leftarrow v_1, v_2$ hyperedge e topologically **do**
  for $z_u \in \mathcal{L}_u$ **do**
    $[\alpha_u]_{z_u} \stackrel{+}{\leftarrow} \sum_{z_1,z_2} [\Psi_e]_{z_u,(z_1,z_2)} \cdot [\alpha_{v_1}]_{z_1} [\alpha_{v_2}]_{z_2}$
return $\sum_z [\alpha_S]_z$

Algorithm 3 Hypergraph marginalization (Matrix form)

for $u \leftarrow v$ hyperedge e topologically **do**
  $\alpha_u \stackrel{+}{\leftarrow} \Psi_e \beta_v$
return $\alpha_S^\top \mathbf{1}$

examples are instances of the hypergraph algorithm.

Example 4.2.1. Hidden Markov Models (HMM) HMMs are discrete latent sequence models defined by the following generative process: first, a sequence of discrete latent states $z = (z_1, \dots, z_T)$ with state size L are sampled as a Markov chain. Then each state z_t independently emits an observation x_t, i.e.

$$p(x, z) = \prod_{t=1}^{T} p(z_t \mid z_{t-1})\, p(x_t \mid z_t), \tag{4.1}$$

where $p(z_t \mid z_{t-1})$ is the transition distribution, $p(x_t \mid z_t)$ the emission distribution, and $p(z_1 \mid z_0)$ is the initial distribution with distinguished start symbol z_0.

Given a sequence of observations $x = (x_1, \dots, x_n)$ we can compute $p(x) = \sum_z p(x, z)$ using a labeled directed hypergraph, with single-tailed edges, nodes corresponding to state positions, labels corresponding to states, and emissions probabilities incorporated into the scoring matrices Ψ. There are T scoring matrices, $\Psi_t \in \mathbb{R}^{\mathcal{L} \times \mathcal{L}}$, with entries $[\Psi_t]_{z_t, z_{t+1}} = p(z_{t+1}, x_t \mid z_t)$ corresponding to transitions.[4] Algorithm 4 shows

[4]The left-most scoring matrix for the HMM has entries $[\Psi_1]_{z_1,z_2} = p(z_2, x_1 \mid z_1) p(z_1 \mid z_0)$.

Algorithm 4 Hypergraph marginalization for HMMs - Backward

for $t \leftarrow (t+1)$ in right-to-left order **do**
  for $z_{t+1} \in \mathcal{L}$ **do**
    $[\beta_{t+1}]_{z_{t+1}} = [\alpha_{t+1}]_{z_{t+1}}$
  $\alpha_t \stackrel{+}{\leftarrow} \Psi_t \beta_{t+1}$
return $\alpha_0^\top \mathbf{1}$

Algorithm 5 Hypergraph marginalization for PCFGs - CKY

for $(i,k) \leftarrow (i,j),(j,k)$ in span-size order **do**
  for $z_1, z_2 \in \mathcal{L}_{i,j} \times \mathcal{L}_{j,k}$ **do**
    $[\beta_{i,j,k}]_{(z_1,z_2)} = [\alpha_{i,j}]_{z_1} [\alpha_{j,k}]_{z_2}$
  $\alpha_{i,k} \stackrel{+}{\leftarrow} \Psi \beta_{i,j,k}$
return $\alpha_{1,T}^\top \mathbf{1}$

the approach. This requires time $O(TL^2)$ and is identical to the backward algorithm for HMMs.[5] //

Example 4.2.2. Context-Free Grammars (CFG) CFGs are a structured model defined by the 5-tuple $\mathcal{G} = (S, \mathcal{N}, \mathcal{P}, \mathcal{X}, \mathcal{R})$, where S is the distinguished start symbol, $\mathcal{N}$ is a set of nonterminals, $\mathcal{P}$ is a set of preterminals, $\mathcal{X}$ is the token types in the vocabulary, and $\mathcal{R}$ is a set of grammar rules. Production rules for start, nonterminals, and preterminals take the following forms:[6]

$$S \to A, \quad A \in \mathcal{N}; \quad A \to B\ C, \quad B, C \in \mathcal{N} \cup \mathcal{P}; \quad D \to x, \quad D \in \mathcal{P}, x \in \mathcal{X}. \tag{4.2}$$

A probabilistic context-free grammar (PCFG) additionally has a probability measure on the set of rules. To compute $p(x_1, \ldots, x_T)$ with a hypergraph, we create one node

[5] In the case of HMMs, the table vectors α_t correspond to the backward algorithm's β values.

[6] We restrict our attention to grammars in Chomsky normal form.

for each contiguous subspan $[i, k)$ in the sentence. Nodes with $i+1 < k$ have a nonterminal label set $\mathcal{L} = \mathcal{N}$. Nodes with $i+1 = k$ have a preterminal label set $\mathcal{L}_{i,i+1} = \mathcal{P}$. The main scoring matrix is $\Psi \in \mathbb{R}^{\mathcal{L}\times\mathcal{L}^2}$, with entries $[\Psi]_{z_u,(z_1,z_2)} = p(z_1, z_2 \mid z_u)$.[7] Algorithm 5 shows how for every hyperedge we join the scores from the two tail nodes in $\alpha_{i,j}$ and $\alpha_{j,k}$ into joined tail vector $\beta_{i,j,k} \in \mathbb{R}^{\mathcal{L}^2}$. As there are $O(T^3)$ hyperedges and the largest $\mathcal{L}$ is of size $|\mathcal{N}|$, the runtime of the algorithm is $O(T^3|\mathcal{N}|^3)$. This approach is identical to the CKY algorithm. //

Example 4.2.3. Hidden Semi-Markov Models (HSMM) HSMMs are extensions of HMMs that allow for generating a variable-length sequence of observations per state. HSMMs are defined by the following generative process: First, we sample a sequence of discrete latent states $z = (z_1, \cdots, z_K)$ with a first-order Markov model. Next, we sample the length of observations under each state from a Poisson distribution $l_k \sim \text{Poisson}(\lambda_{z_k})$ truncated at max length M. The joint distribution is defined as

$$p(x, z, l) = \prod_{k=1}^{K} p(z_k \mid z_{k-1})\, p(l_k \mid z_k) \prod_{i=l_1+\cdots+l_{k-1}}^{l_1+\cdots+l_k} p(x_i \mid z_k), \tag{4.3}$$

where the sequence length T can be computed as $T = \sum_{k=1}^{K} l_k$. In this work, we only consider modeling continuous x_t, so we use a Gaussian distribution for $p(x_i \mid z_k)$.

To compute $p(x)$, we marginalize over l, z using dynamic programming similar to HMMs, except that we have an additional factor of M: the overall complexity is $O(T \times M \times L^2)$ (ignoring the emission part since they are usually not the bottleneck). We refer to Yu (2010) for more details. //

[7] We have a separate matrix for terminal production on x which we elide for simplicity.

4.3 Rank-Constrained Structured Models

For these structured distributions, hypergraphs provide a general method for inference (and therefore training parameterized versions). However, the underlying algorithms scale poorly with the size of the label sets (quadratic for HMM and HSMM, cubic for CFG). This complexity makes it challenging to scale these models and train versions with very large numbers of states.

In this section, we consider an approach for improving the scalability of these models by reducing the dependence of the computational complexity of inference on the label set size. The main idea is to speed up the matrix-vector product step in inference by using a low-rank decomposition of the scoring matrix Ψ. In the next section we show that this constraint can be easily incorporated into parameterized versions of these models.

4.3.1 Low-Rank Matrix-Vector Products

The main bottleneck for inference speed is the matrix-vector product $\alpha_u \overset{+}{\leftarrow} \Psi_e \beta_v$ that must be computed for every edge in the hypergraph. As we saw in Algorithm 3, this step takes time $L^{|e|+1}$ to compute, but it can be sped up by making structural assumptions on Ψ_e. In particular, we focus on scoring matrices with low rank.

We note the following elementary property of matrix-vector products. If the scoring matrix can be decomposed as the product of two smaller matrices $\Psi_e = U_e V_e^\top$, where $U_e \in \mathbb{R}^{\mathcal{L} \times N}$ and $V_e \in \mathbb{R}^{N \times \mathcal{L}^{|e|}}$, then the matrix-vector products can be computed in time $O(|\mathcal{E}| \times L^{|e|} \times N)$ as follows:

$$\Psi_e \beta_v = \left(U_e V_e^\top\right) \beta_v = U_e \left(V_e^\top \beta_v\right). \tag{4.4}$$

Algorithm 6 Low-rank marginalization

for $u \leftarrow v_1, v_2$ hyperedge e topologically **do**
  for $n \in 1, \dots, N$ **do**
    $[\gamma]_n = \sum_{z_v} c_v \, [\phi(g(z_1, z_2))]_n \, [\beta_v]_{z_v}$ $\triangleright \; O(L^{|e|})$
  $\alpha_u \overset{+}{\leftarrow} U_e \gamma$ $\triangleright \; O(LN)$
return $\alpha_S^\top \mathbf{1}$

This reordering of computation exchanges a factor of L for a factor of N. When $N \ll L$, this method is both faster and more memory-efficient.

We enforce the low-rank constraint by directly parameterizing the factors U_e and V_e for scoring matrices Ψ_e that we would like to constrain. We treat both U_e and V_e as embedding matrices, where each row corresponds to an embedding of each value of z_u and a joint embedding of (z_1, z_2) respectively:

$$[U_e]_{z_u,n} = c_{z_u}[\phi(f(z_u))]_n \qquad [V_e]_{(z_1,z_2),n} = c_{z_1,z_2}[\phi(g(z_1, z_2))]_n, \tag{4.5}$$

where f and g are embedding functions; c_{z_u} and c_{z_1,z_2} are constants (used to ensure proper normalization) or clamped potentials (such as conditional probabilities); and $\phi : \mathbb{R}^D \to \mathbb{R}_+^N$ is a function that ensures nonnegativity, necessary for valid probability mass functions. Algorithm 6 shows the role of the low-rank matrix-vector product in marginalization.[8]

[8] If the normalizing constants are given by c_{z_u}, they can be computed from unnormalized $\tilde{U}_e, \tilde{V}_e$ as follows: $c_{z_u} = [\tilde{U}_e \tilde{V}_e^\top \mathbf{1}]_{z_u}$ in time $O(L^{|e|}N + LN)$, and similarly for c_{z_1,z_2}.

4.3.2 Application to Structured Models

As enforcing a low-rank factorization of every scoring matrix limits the expressivity of a model, we explicitly target scoring matrices that are involved in computational bottlenecks.[9] For these key scoring matrices, we directly parameterize the scoring matrix with a low-rank factorization, which we call a low-rank parameterization. For other computations, we utilize a standard softmax parameterization and do not factorize the resulting scoring matrix. We refer to this as a mixed parameterization.

Example 4.3.1. Hidden Markov Models Low-rank HMMs (LHMMs) use the following mixed parameterization, which specifically targets the state-state transition bottleneck by using a low-rank parameterization for the transition distribution, but a softmax parameterization for the emission distribution:

$$\begin{aligned} p(z_1 \mid z_0) &\propto \phi(f_1(\mathbf{u}_{z_0}))^\top \phi(\mathbf{v}_{z_1}) \\ p(z_t \mid z_{t-1}) &\propto \phi(\mathbf{u}_{z_{t-1}})^\top \phi(\mathbf{v}_{z_t}) \\ p(x_t \mid z_t) &\propto \exp(\mathbf{u}_{z_t}^\top f_2(\mathbf{v}_{x_t})), \end{aligned} \tag{4.6}$$

where $\mathbf{u}_z$ is the embedding of z when z is used as head, $\mathbf{v}_z$ its embedding when used as tail, f_1, f_2 are MLPs with two residual layers, and feature map $\phi(x) = \exp(Wx)$. The MLPs are parameterized following Kim et al. (2019):

$$\begin{aligned} f_i(x) &= g_{i,1}(g_{i,2}(W_i x)), \\ g_{i,j}(y) &= \mathrm{ReLU}(U_{i,j}\mathrm{ReLU}(V_{i,j} y)) + y, \end{aligned} \tag{4.7}$$

with $i, j \in \{1, 2\}$, and $W_i, V_{i,j}, U_{i,j} \in \mathbb{R}^{D \times D}$.

[9] For a discussion of the expressivity of low-rank models compared to models with fewer labels, we refer the reader to Appendix A of Chiu et al. (2021).

When performing inference, we treat the emission probabilities $p(x_t \mid z_t)$ as constants, and absorb them into c_u. This allows inference to be run in time $O(TLN)$, where T is the length of a sequence, L the size of the label space, and N the feature dimension. //

Example 4.3.2. Hidden Semi-Markov Models For low-rank HSMM (LHSMM), we similarly target the transition distribution and keep the standard Gaussian emission distribution:

$$p(z_k \mid z_{k-1}) \propto \mathbf{u}_{z_{k-1}}^\top \mathbf{v}_{z_k}, \quad p(x_t \mid z_k) \propto K_{\text{Gauss}}(\mathbf{u}_{z_k}, \mathbf{x}_t), \tag{4.8}$$

where $\mathbf{u}_{z_{k-1}} = \phi(f(z_{k-1}))$ and $\mathbf{v}_{z_k} = \phi(g(z_k))$ are state embeddings, while $K_{\text{Gauss}}(\cdot, \cdot)$ is the Gaussian kernel used to model continuous $\mathbf{x}_t$. The total inference complexity is $O(TLMN)$, where M is the maximum length of the observation sequence under any state. //

Example 4.3.3. Context-Free Grammars For PCFGs, the inference bottleneck is related to the transition from a nonterminal symbol to two nonterminal symbolss ($A \to B\ C$), and we specifically parameterize it using a low-rank parameterization:

$$p(z_{1,N} \mid S) \propto \exp(\mathbf{u}_S^\top \mathbf{u}_{z_{1,N}}), \quad p(z_{i,j}, z_{j,k} \mid z_{i,k}) \propto \begin{cases} \exp(\mathbf{u}_{z_{i,k}}^\top \mathbf{v}_{z_{i,j}\ z_{j,k}}) & \substack{i+1=j \vee \\ j+1=k} \\ \phi(\mathbf{u}'_{z_{i,k}})^\top \phi(\mathbf{v}_{z_{i,j}}),_{z_{j,k}} & \text{o.w.} \end{cases} \tag{4.9}$$
$$p(x_i \mid z_i) \propto \exp(\mathbf{u}_{z_i}^\top \mathbf{v}_{x_i}),$$

where $\mathbf{u}_z/\mathbf{u}'_z$ is the embedding of z when z is used as head,$\mathbf{v}_x/\mathbf{v}_{z_1,z_2}$ is the embedding of $x/(z_1, z_2)$ when they are used as tail. We use the same MLP parameterization of z as in LHMMs, drawn from Kim et al. (2019). Note that we limit the application of

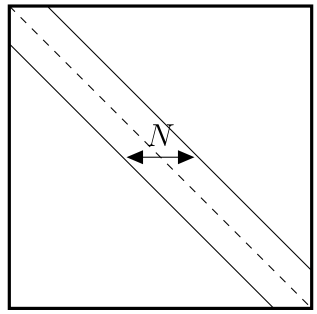


Figure 4.1: An example of a banded matrix with width N, which has $N/2$ nonzero elements on both sides of the diagonal for each row.

low-rank constraints to nonterminal to nonterminal productions. These productions dominate the runtime as they are applied at $O(T^3)$ hyperedges. This allows inference to be run in time $O(T^3 L^2 N)$, where T is the length of a sequence, L the size of the label space, and N the feature dimension. //

4.4 Low-rank and Banded HMM Parameterization

The low-rank constraint may be too restrictive in some settings. For example, a low-rank model is unable to fit the identity matrix, which would have rank L. To overcome this limitation, we extend the low-rank model while preserving the computational complexity of inference. We add a set of parameters $\theta \in \mathbb{R}^{L \times L}$ which allow the model to learn high-rank structure. We constrain θ to have banded structure, such that $[\theta]_{z_{t-1}, z_t} = 0$ if $|z_t - z_{t-1}| > N/2$. See Figure 4.1 for an illustration.

With the addition of banded structure, training and inference remain efficient. We first give the full parameterization of banded and low-rank HMMs, then show the complexity of training is unchanged. Let band segment $B_z = \{z' : |z - z'| \leq N/2\}$.

The transition probabilities are then given by

$$p(z_t \mid z_{t-1}) = \frac{[\theta]_{z_{t-1},z_t} + \phi(\mathbf{u}_{z_{t-1}})^\top \phi(\mathbf{v}_{z_t})}{Z_{z_{t-1}}}, \tag{4.10}$$

with normalizing constants

$$\begin{aligned} Z_{z_{t-1}} &= \sum_{z_t} [\theta]_{z_{t-1},z_t} + \phi(\mathbf{u}_{z_{t-1}})^\top \phi(\mathbf{v}_{z_t}) \\ &= \sum_{z_t \in B_{z_{t-1}}} [\theta]_{z_{t-1},z_t} + \phi(\mathbf{u}_{z_{t-1}})^\top \sum_{z_t} \phi(\mathbf{v}_{z_t}). \end{aligned} \tag{4.11}$$

The normalization constant for each starting state $Z_{z_{t-1}}$ can be computed in time $O(N)$.

This allows us to perform inference quickly. We can use the above to rewrite the score matrix $\Psi_t \propto \theta + UV^\top$, which turns the inner loop of Eqn. 4.4 (specialized to HMMs) into

$$\alpha_t = \Psi_t \beta_{t+1} \propto (\theta + UV^\top)\beta_{t+1} = \theta\beta_{t+1} + U(V^\top \beta_{t+1}), \tag{4.12}$$

omitting constants (i.e. emission probabilities and normalizing constants). Since θ is banded, the banded matrix-vector product $\theta\beta_t$ takes time $O(LN)$. This update, in combination with the low-rank product, takes $O(LN)$ time total. Each update in the hypergraph marginalization algorithm is now 3 matrix-vector products costing $O(LN)$ each, preserving the runtime of inference.

4.5 Experimental Setup

We evaluate the application of low-rank constraints with four experiments: sequential language modeling with HMMs, polyphonic music modeling with a large observation space, hierarchical language modelings with PCFGs, and video modeling with HSMMs.

Data Our first set of experiments evaluate sequential models on Penn Treebank dataset (Ptb) (Marcus et al., 1993) for the task of word-level language modeling. We use the preprocessing from Mikolov et al. (2011). The second set of experiments is on polyphonic music modeling (Boulanger-Lewandowski et al., 2012). We evaluate on four music datasets: Nottingham (Nott), Piano, MuseData (Muse), and JSB chorales (JSB). Each timestep consists of an 88-dimensional binary vector indicating whether a particular note is played. Since multiple notes may be played at the same time, the effective vocabulary size is extremely large. The third set of experiments use PCFGs for language modeling, we also use Ptb, but with the splits and preprocessing used in unsupervised constituency parsing (Shen et al., 2018, 2019, Kim et al., 2019). The last set of experiments use HSMMs for video modeling, where we use CrossTask (Zhukov et al., 2019) with 10% of the training data for validation. We follow the preprocessing steps in Fried et al. (2020) and apply PCA to project features to vectors of size 200.

Training details For language modeling with HMMs, we experiment with a range of state sizes, $|\mathcal{L}| = L \in \{2^{10}, 2^{11}, 2^{12}, 2^{13}, 2^{14}\}$, and rank $N \in \{L/2, L/4, L/8\}$. For polyphonic music modeling with HMMs, we experiment with states sizes $L \in \{2^7, 2^8, 2^9, 2^{10}, 2^{11}\}$. For language modeling with PCFGs, we use a set of nonterminals of size $|\mathcal{N}| \in \{30, 60, 100\}$ and preterminals of twice the number of nonterminals $|\mathcal{P}| = 2|\mathcal{N}|$. Our smallest setting ($|\mathcal{N}| = 30$, $|\mathcal{P}| = 60$) is the one used in Kim et al. (2019). For video modeling with HSMMs, we use the same model setting as Fried et al. (2020), but we don't constrain states to the predefined states per task, and we experiment with state sizes $L \in \{2^6, 2^7, 2^8, 2^9, 2^{10}\}$ and rank $N \in \{2^4, 2^5, 2^6, 2^7\}$.

We utilize the feature map $\phi(x) = \exp(Wx)$ for the LHMM and LHSMM, and $\phi(x) = \exp(Wx - \|x\|_2^2/2)$ for the LPCFG. We initialize the parameters of feature maps using orthogonal feature projections (Choromanski et al., 2020), and update it alongside the model parameters.

We initialize the parameters W, in $\phi(x) = \exp(Wx)$ and variants, of feature maps using orthogonal feature projections (Choromanski et al., 2020), and update it alongside the model parameters during training.

HMM parameters are initialized with the Xavier initialization (Glorot & Bengio, 2010).[10] We use the AdamW (Loshchilov & Hutter, 2017) optimizer with a learning rate of 0.001, $\beta_1 = 0.9, \beta_2 = 0.999$, weight decay 0.01, and a max grad norm of 5. We use a state dropout rate of 0.1, and additionally have a dropout rate of 0.1 on the feature space of LHMMs. We train for 30 epochs with a max batch size of 256 tokens, and anneal the learning rate by dividing by 4 if the validation perplexity fails to improve after 4 evaluations. Evaluations are performed 4 times per epoch. The sentences, which we model independently from one another, are shuffled after every epoch. Batches of sentences are drawn from buckets containing sentences of similar lengths to minimize padding.

For the polyphonic music datasets, we use the same hyperparameters as the lan-

[10]For banded experiments, we initialize the band parameters by additionally adding 30 to each element. Without this the band scores were too small compared to the exponentiated scores, and were ignored by the model.

guage modeling experiments, except a state dropout rate of 0.5 for JSB and Nottingham, 0.1 for Muse and Piano. We did not use feature space dropout in the LHMMs on the music datasets. For Nottingham and JSB, sentences were batched in length buckets, the same as language modeling. Due to memory constraints, Muse and Piano were processed using BPTT with a batch size of 8 for Muse and 2 for Piano, and a BPTT length of 128. We use $D = 256$ for all embeddings and MLPs on all datasets, except Piano, which due to its small size required $D = 64$ dimensional embeddings and MLPs.

For PCFGs, parameters are initialized with the Xavier uniform initialization (Glorot & Bengio, 2010). We follow the experiment setting in Kim et al. (2019) and use the Adam (Kingma & Ba, 2017) optimizer with $\beta_1 = 0.75, \beta_2 = 0.999$, a max grad norm of 3, and we tune the learning rate from $\{0.001, 0.002\}$ using validation perplexity. We train for 15 epochs with a batch size of 4. The learning rate is not annealed over training, but a curriculum learning approach is applied where only sentences of at most length 30 are considered in the first epoch. In each of the following epochs, the longest length of sentences considered is increased by 1.

For HSMMs, we use the same initialization and optimization hyperparameters as Fried et al. (2020): The Gaussian means and covariances are initialized with empirical means and covariances (the Gaussian parameters for all states are initialized the same way and they only diverge through training). The transition matrix is initialized to be uniform distribution for baseline HSMMs, and the transition embeddings are initialized using the Xavier initialization for LHSMMs. The log of Poisson parameters are initialized to be 0. We train all models for 4 epochs using the Adam optimizer with initial learning rate of 5e-3, and we reduce the learning rate 80% when log likelihood doesn't improve over the previous epoch. We clamp the learning rate to be at least 1e-

4. We use a batch size of 5 following Fried et al. (2020), simulated by accumulating gradients under batch size 1 in order to scale up the number of states as much as we can. Gradient norms are clipped to be at most 10 before updating. Training takes 1-2 days depending on the number of states and whether a low-rank constraint is used.

We use the following hardware for our experiments: for HMMs we run experiments on 8 Titan RTX GPUs with 24G of memory on an internal cluster. For PCFGs and HSMMs we run experiments on 1 Nvidia V100 GPU with 32G of memory on an internal cluster.

Baselines and Evaluation The language modeling experiments are evaluated using perplexity. Baselines are neurally parameterized HMM with a standard softmax transition. We also compare to VL-HMM, which makes a strong structural sparsity assumption on the emission distribution (Chiu & Rush, 2020b). We include for reference a state-of-the-art language model, the AWD-LSTM (Merity et al., 2017). For polyphonic music modeling, we compare our LHMM against RNN-NADE (Boulanger-Lewandowski et al., 2012) which models the full joint distribution of notes as well as temporal dependencies; as well as autoregressive neural models such as the R-Transformer (Wang et al., 2019) (as reported by Song et al. (2019)) and an LSTM (as reported by Ziegler & Rush (2019)); models with latent continuous dynamics such as the LV-RNN (Gu et al., 2015) and SRNN (Fraccaro et al., 2016); and finally comparable models with latent discrete dynamics, the TSBN (Gan et al., 2015) and the baseline HMM. We evaluate perplexities of our low-rank PCFG (LPCFG) against a softmax PCFG (PCFG) (Kim et al., 2019). For video modeling, we evaluate the negative log likelihoods on the test set and compare low-rank HSMMs to softmax HSMMs.

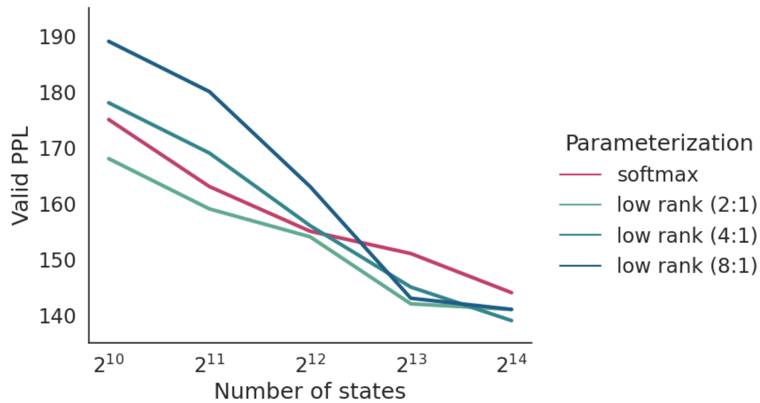


Figure 4.2: Validation perplexities on PTB versus model scale.

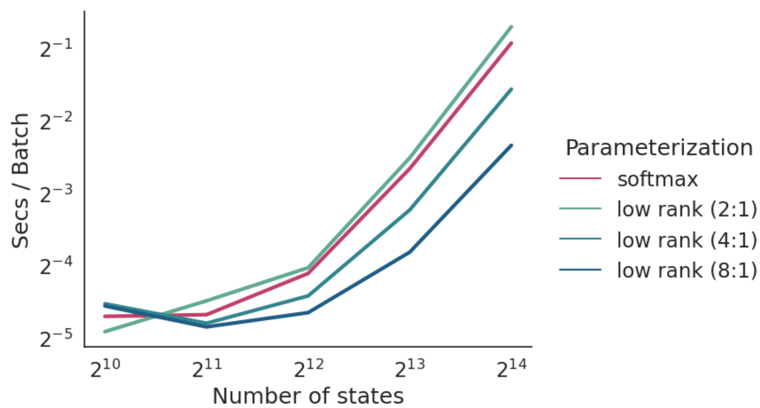


Figure 4.3: Model speed in seconds per batch.

4.6 Results

Hidden Markov Models for Language Modeling Our main experimental result is that the low-rank models achieve similar accuracy, as measured by perplexity, as our baselines. Fig. 4.2 shows that perplexity improves as we increase the scale of the HMM, and that the performance of our LHMM also improves at the same rate. At small sizes, the low-rank constraints slightly hinder accuracy; however once the size is large enough, i.e. larger than 2^{12}, LHMMs with 8:1 state-to-rank ratios perform com-

Model	Val	Test
AWD-LSTM	60.0	57.3
VL-HMM	128.6	119.5
HMM	144.3	136.8
LHMM	141.4	131.8

Table 4.1: Model perplexities on PTB. All HMM variants have $L = 2^{14}$ states. Validation and test perplexities. The LHMM has a state-to-rank ratio $8 : 1$.

Model	$L : N$	Train	Val
HMM	-	95.9	144.3
LHMM	8	97.5	141.4
LHMM+band	8	101.1	143.8
LHMM	16	110.6	146.3
LHMM+band	16	96.9	138.8
LHMM	32	108.4	153.7
LHMM+band	32	110.7	145.0

Table 4.2: Model perplexities on PTB. All HMM variants have $L = 2^{14}$ states. (Left): Validation and test perplexities. The LHMM has a state-to-rank ratio $8 : 1$. (Right): Further experiments with extending the low-rank structure of LHMMs with a banded transition structure.

parably.

Fig. 4.2 also contains speed comparisons between HMMs and LHMMs. A state-to-rank ratio of 8:1 matches the accuracy of softmax HMMs at larger state sizes and also gives an empirical speedup of more than 3x at $L = 2^{14}$. As expected, we only see a speedup when the state-to-rank ratio exceeds 2:1, as we replaced the $O(L^2)$ operation with two $O(LN)$ ones. This implies that the low-rank constraint is most effective with scale, where we observe large computational gains at no cost in accuracy.

HMMs are outperformed by neural models, and also by VL-HMMs (Chiu & Rush, 2020b) which offer similar modeling advantages to HMMs, as shown in Tbl. 4.2 (left). This indicates that some aspects of performance are not strictly tied to scale. We posit

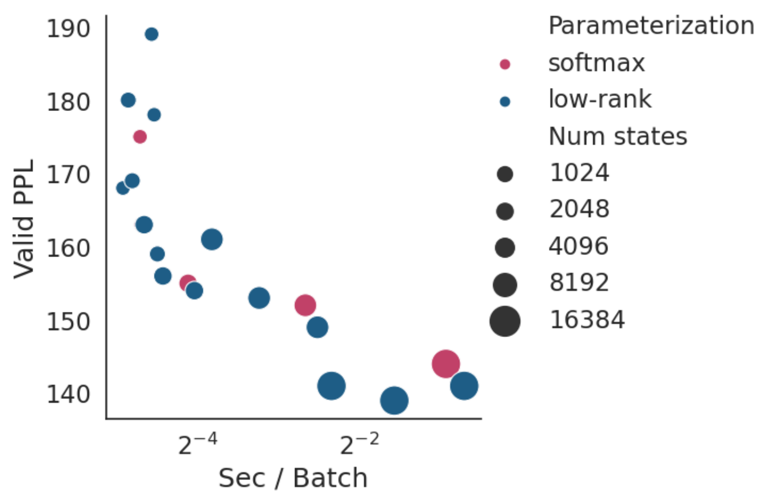


Figure 4.4: The speed, in seconds per batch, versus accuracy, in perplexity, for HMMs and low-rank versions over a range of model sizes. As lower is better for both measures of speed and accuracy, the frontier is the bottom left.

this is due to the problem-specific block-sparse emission constraint in VL-HMMs. While very effective for language modeling, the VL-HMM relies on a hard clustering of states for constraining emissions. This is difficult to apply to problems with richer emission models (as in music and video modeling).

We also perform a rank analysis of the trained HMMs and LHMMs, shown in Table 4.3. We estimate rank by counting the number of singular values greater than 1e-5. Note that the feature dimension N is the maximum attainable rank for the transition matrix of an LHMM. Although LHMMs often manage to achieve the same validation perplexity as HMMs at relatively small N, the ranks of the transition matrices are much lower than both their HMM counterparts as well as N. At larger state sizes, the ranks of learned matrices are almost half of their max achievable rank. Interestingly, this holds true for HMMs as well, with the empirical rank of the transition matrices significantly smaller than the number of states.

Model	L	N	rank(A)	rank(O)	Val PPL
HMM	16384	-	9187	9107	144
LHMM	16384	8192	2572	7487	141
LHMM	16384	4096	2016	7139	144
LHMM	16384	2048	1559	6509	141
LMM	8192	-	5330	5349	152
LHMM	8192	4096	1604	5113	149
LHMM	8192	2048	1020	4980	153
LHMM	8192	1024	791	5033	161
HMM	4096	-	2992	3388	155
LHMM	4096	2048	1171	3300	154
LHMM	4096	1024	790	2940	156
LHMM	4096	512	507	3186	163

Table 4.3: Ranks and validation perplexities for HMMs and LHMMs. The number of states is given by L and the dimensionality of the feature space by N. The HMM uses softmax for the emission, and therefore does not have a value for N. The transition matrix is denoted by A, and the emission matrix by O. The rank was estimated by counting the number of singular values greater than 1e-5. Models were trained with 0.1 state and feature dropout.

One potential limitation of all low-rank models is that they cannot learn high-rank structures with low N. We began to see this issue at a ratio of 16:1 states to features for large HMMs. To explore the effects of this limitation, we perform an additional experiment that combines low-rank features with a sparse component. Specifically, we add an efficient high-rank sparse banded transition matrix. Table 4.2 shows that combination with the band structure allows for larger ratios than just the low-rank structure alone, while only adding another operation that costs $O(LN)$.

Hidden Markov Models for Music Modeling We next apply LHMMs on polyphonic music modeling. This has a max effective vocabulary size of 2^{88}, as multiple notes may occur simultaneously. Unlike for language modeling, we use a factored

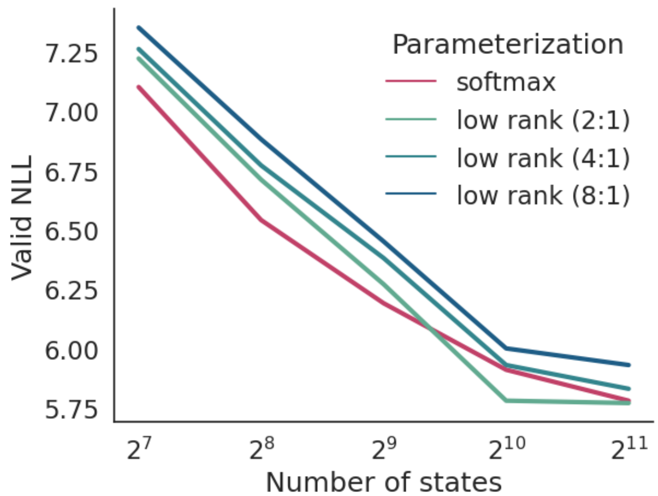


Figure 4.5: Polyphonic music negative log-likelihoods (NLL), measured in nats. HMM and LHMM validation performance for various state sizes and state:rank ratios.

Bernoulli emission model, modeling the presence of each note independently. Fig. 4.6 (right) shows that HMMs are competitive with many of the models on these datasets, including LSTMs. We find that LHMMs achieve performance slightly worse than but comparable to the unconstrained HMMs overall. Fig. 4.6 (left) shows that the distinction drops with more states. Both HMMs achieve low negative likelihoods (NLL) on the datasets with shorter sequences, Nottingham and JSB, but relatively poorer NLLs on the datasets with longer sequences (Muse and Piano).

Context-Free Grammars For syntactic language modeling on Ptb, our low-rank PCFG (LPCFG) achieves similar performance to PCFGs, as shown in Table 4.4 (left), with an improvement in computational complexity. The complexity of inference in PCFGs models is cubic in the number of nonterminals, so even models with $|\mathcal{N}| = 30$ nonterminals are relatively costly. Our approach achieves comparable results with $N = 8$ features. As we scale up the number of nonterminals to $|\mathcal{N}| = 100$, LPCFG

Model	Nott	Piano	Muse	JSB
RNN-NADE	2.31	**7.05**	**5.6**	5.19
R-Transformer	**2.24**	7.44	7.00	8.26
LSTM	3.43	7.77	7.23	8.17
LV-RNN	2.72	7.61	6.89	**3.99**
SRNN	2.94	8.20	6.28	4.74
TSBN	3.67	7.89	6.81	7.48
HMM	2.43	8.51	7.34	5.74
LHMM	2.60	8.89	7.60	5.80

Figure 4.6: Polyphonic music negative log-likelihoods (NLL), measured in nats. (Left): HMM and LHMM validation performance for various state sizes and state:rank ratios. (Right): Test-set NLLs for polyphonic music. The HMM models have $\mathcal{L} = 2^{11}$ states and the LHMM has rank $N = 2^9$, a 4:1 state:rank ratio.

$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{P}\rvert$	Model	N	PPL	Secs
30	60	PCFG	-	252.60	0.23
		LPCFG	8	247.02	0.27
		LPCFG	16	250.59	0.27
60	120	PCFG	-	234.01	0.33
		LPCFG	16	217.24	0.28
		LPCFG	32	213.81	0.30
100	200	PCFG	-	191.08	1.02
		LPCFG	32	203.47	0.64
		LPCFG	64	194.25	0.81

Table 4.4: Test perplexities and speeds for PCFG models on Ptb. The complexity of PCFG is $O(T^3|\mathcal{N}|^3)$, whereas the complexity of LPCFG is $O(T^3|\mathcal{N}|^2N)$. Speeds are given in seconds per batch.

stays competitive with a lower computational complexity (since $N < |\mathcal{N}|$). These experiments also demonstrate the importance of scale in syntactic language models with more than 50 point gain in perplexity over a strong starting model.

We also analyze the speed of PCFGs in Table 4.4. Once the model is large enough,

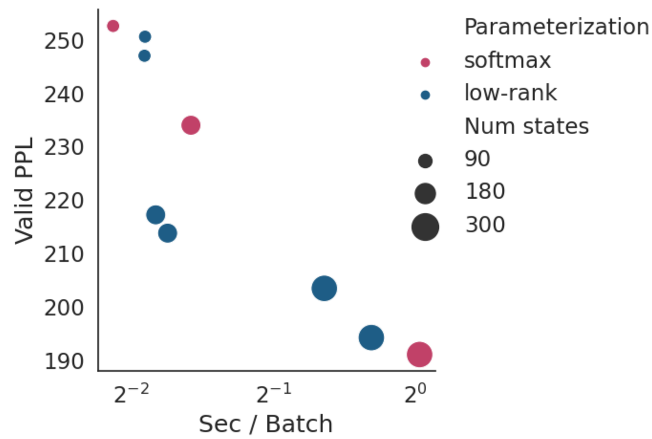


Figure 4.7: The speed, in seconds per batch, versus accuracy, in perplexity, for PCFGs and low-rank versions over a range of model sizes. As lower is better for both measures of speed and accuracy, the frontier is the bottom left.

i.e. $|\mathcal{N}| \geq 60$ nonterminals and $|\mathcal{P}| \geq 120$ preterminals, the LPCFG is faster than PCFG. Note that the LPCFG is faster than the CFG even when the number of features $N > \frac{|\mathcal{N}|}{2}$, in contrast to the HMM case where a speedup can only be obtained when $N < L/2$. This is due to the scoring matrix being rectangular: Recall the low-rank matrix product $\Psi\beta = U(V^\top\beta)$, where, when specialized to PCFGs, the left-hand side takes time $O(L^3)$ and the right-hand side takes $O(L^2N + LN)$. For PCFGs, the term $V^\top\beta$ dominates the runtime. This contrasts with HMMs, where both $V^\top\beta$ and the subsequent multiplication by U take the same amount of time, $O(LN)$. We show the frontier in Figure 4.7.

Hidden Semi-Markov Models for Video Modeling Table 4.4 (right) shows the results of video modeling using HSMMs. In addition to using a different hypergraph for inference, these experiments use a continuous Gaussian emission model. By removing the state constraints from tasks, our HSMM baselines get better video-level NLLs than that from Fried et al. (2020) at the cost of more memory consumption.

Model	L	N	NLL	Secs
HSMM[11]	151	-	$1.432e5$	-
HSMM	2^6	-	$1.428e5$	0.78
HSMM	2^7	-	$1.427e5$	2.22
HSMM	2^8	-	$1.426e5$	7.69
LHSMM	2^7	2^7	$1.427e5$	4.17
LHSMM	2^8	2^6	$1.426e5$	5.00
LHSMM	2^9	2^5	$1.424e5$	5.56
LHSMM	2^{10}	2^4	$1.423e5$	10.0

Table 4.5: Negative log likelihoods (NLL) per video and speeds for HSMM models on CrossTask. We cannot train HSMMs beyond 2^8 states due to GPU memory constraints, but we can train LHSMMs with up to 2^{10} states. Speeds are given in seconds per batch.

Due to GPU memory constraints, we can only train HSMMs up to 2^8 states. However, the low-rank parameterization allows models to scale to 2^{10} states, yielding an improvement in NLL. Absolute results could likely be improved with more states and by an improved emission parameterization for all models.

4.7 Related Work

Similar to our work, other approaches target matrix or tensor operations in inference, and impose structural model constraints to improve computational complexity. Many of the works on HMMs in particular take advantage of the transition structure. The Dense-mostly-constant (DMC) HMM assigns a subset of learnable parameters per row of the transition matrix and sets the rest to a constant, leading to a sub-quadratic runtime (Siddiqi & Moore, 2005). Other structures have also been explored, such as aligning the states of an HMM to underlying phenomena that allows inference to be sped up (Felzenszwalb et al., 2004, Roweis, 2000). Additionally, other methods take advantage of emission structure in HMMs in order to scale, such as the Cloned HMM (Dedieu et al., 2019) and VL-HMM (Chiu & Rush, 2020b). Compared to these approaches, our method is more flexible and generic, since it can be applied in a non-application-specific manner, and even extended with high-rank components (such as banded structure).

Low-rank structure has been explored in both HMMs (Siddiqi et al., 2009), a generalization of PCFGs called weighted tree automata (Rabusseau et al., 2015), and conditional random fields (Thai et al., 2018). The reduced-rank HMM (Siddiqi et al., 2009) has at most 50 states, and relies on spectral methods for training. The low-rank weighted tree automata (Rabusseau et al., 2015) also trains latent tree models via spectral methods. We extend the low-rank assumption to neural parameterizations, which have been shown to be effective for generalization (Kim et al., 2019, Chiu & Rush, 2020b), and directly optimize the evidence via gradient descent. Finally, Thai et al. (2018) do not take advantage of the low-rank parameterization of their CRF potentials for faster inference via low-rank matrix products, a missed opportunity. Instead, the low-rank parameterization is used only as a regularizer, with the full potentials instantiated during inference.

Concurrent work in unsupervised parsing uses a tensor decomposition to scale PCFGs to large state spaces (Yang et al., 2021b). Our low-rank decomposition of the flattened head-tail scoring matrix is more general, resulting in worse scaling for the PCFG setting but with wider applicability, as shown by experiments with HMMs and HSMMs.

4.8 Conclusion

Our approach improves the scaling of structured, discrete state space models by establishing the effectiveness of low-rank constraints for hypergraph models. We show that viewing a key step of inference in structured models as a matrix-vector product, in combination with a low-rank constraint on relevant parameters, allows for an immediate speedup. Low-rank inference allows us to obtain a reduction in the asymptotic complexity of marginalization at the cost of a constrained model. Our approach applies to a wide class of models, including HMMs, HSMMs, and PCFGs. Through our experiments on language, video, and polyphonic music modeling, we demonstrate an effective approach for overcoming the practical difficulty of applying low-rank constraints in high dimensional, structured spaces by targeting and constraining model components that bottleneck computation.

An important avenue of future work is to explore constraints that allow models with fewer independence assumptions to scale, as well as how to automate the design of interpretable models. The class of models explored in this chapter should be extended to more accurate and controllable models, as modern autoregressive language models are accurate but difficult to control.

This chapter concludes our exploration of language modeling. In this chapter and the last, we show how neural parameterizations coupled with structural constraints lead to scalable and accurate state tracking with explicit uncertainty over the state. In the rest of the thesis, we will explore state tracking representations in two more applications: question answering and dialogue.

5

HOP, UNION, GENERATE: Explainable Multi-hop Reasoning without Rationale Supervision

5.1 Introduction

The next application we consider is question answering (QA), where we explore a state tracking representation for multi-hop reasoning. Multi-hop reasoning is central to question answering. It allows agents to answer questions by retrieving a sequence of facts that, when composed, lead to the correct answer. When retrieving facts, agents must keep track of the already retrieved facts to reason about what information is missing. This chapter proposes a new discrete state tracking representation tailored to structured information selection for multi-hop reasoning.

In this chapter, we refer to the structured state representation as a *rationale* – the set of sentences that lead to the answer. Depending on task specifications, the rationale can be within a single document, or span across multiple documents. Explainable multi-hop QA has been extensively studied in the supervised setting, where both rationale annotations and answer annotations are given. These approaches either apply multi-task loss functions (Joshi et al., 2020, Groeneveld et al., 2020, DeYoung et al., 2020) or design specialized network architectures (Tu et al., 2019, Fang et al., 2020). However, having access to rationale annotations is a strong assumption. In practice, they are expensive to collect (Geva et al., 2021), less available than answer annotations (Welbl et al., 2018), and can suffer from low agreement rates between annotators (Zhang et al., 2020).

Researchers have thus explored approaches that do not require rationale annotations (Lewis et al., 2020b, Glockner et al., 2020, Atanasova et al., 2022). However, these previous approaches limit their reasoning to information from 1 or 2 sentences, and so they cannot be applied in *multi-hop* scenarios, i.e. QA tasks that require making connections between several pieces of information across sentences and across documents. Additionally, these methods are either restricted to only work for multiple-choice QA, or restricted to only produce rationales at the document level but not at the sentence level.

We propose Hop, Union, Generate (HUG), a principled, probabilistic approach for training explainable multi-hop QA systems without rationale supervision. HUG overcomes the two-sentence limitation of previous methods by directly reasoning about rationales as *sets* of sentences, while also extending rationale prediction to the multi-document setting. We show an overview of HUG in Figure 5.1. HUG leverages the naturally hierarchical structure of text and proceeds in three stages – it first selects the relevant set of documents given the question (Hop); then, it selects a subset of sentences within each of the relevant documents and collects them together (Union); finally, it generates an answer via a seq2seq model with all the collected sentences (Generate). The key to multi-hop reasoning in HUG is modeling each selection as an explicit distribution over sets. A probabilistic set distribution compares non-contiguous and variable size rationales, affording HUG flexibility for rationale selection.

Training a set-prediction model quickly becomes intractable as the size increases. We make two algorithmic choices that lead to tractable training for HUG. Treating rationales as a latent variable requires HUG to marginalize over all possible rationales, leading to an intractable learning objective. HUG overcomes this issue by performing sampling in a hierarchical way – it first identifies the most promising documents and then the most promising sentences within those documents. Second, multi-hop QA often involves reasoning over long documents, which is challenging due to the computational complexity of encoding long documents with neural models such as transformers. To make this encoding efficient, HUG performs computation in the embedding space.

We empirically evaluate HUG on three different multi-hop QA datasets: HotpotQA (Yang et al., 2018), MuSiQue (Trivedi et al., 2022), FEVER (DeYoung et al., 2020), and MultiRC (DeYoung et al., 2020). Results show that, for both selecting rationales and predicting answers, HUG is better than a number of state-of-the-art semi-supervised and unsupervised methods (Chen et al., 2019, Lewis et al., 2020b, Glockner et al., 2020, Atanasova et al., 2022) on all of the datasets. We also demonstrate that HUG com-

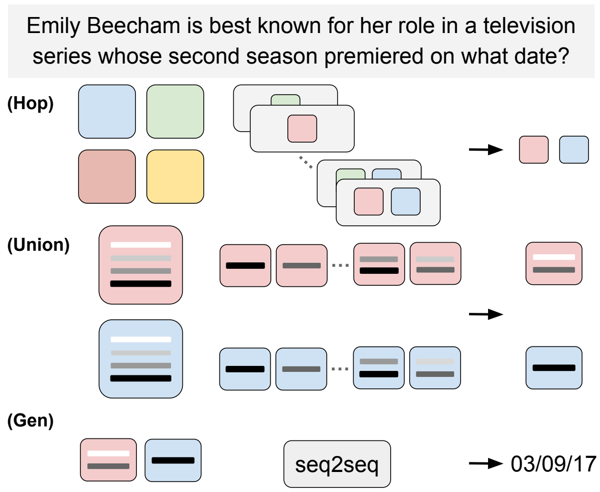


Figure 5.1: An overview of HUG, which proceeds in three stages. **Hop** explicitly considers all possible document sets and selects the most likely document set, **Union** explicitly considers all sentence subsets and chooses the most likely sentence subset within each selected document, and **Generate** combines the chosen sentence subsets and generates an answer.

bined with larger language models consistently produces better performance. We then analyze performance according to different types of multi-hop reasoning and show that by explicitly modeling multi-hop reasoning, HUG achieves a large improvement on the reasoning type that requires bridging entities.

5.2 Generative Multi-Hop QA

In the standard multi-hop QA setting, an example consists of a question x, a set of documents D, and an answer y. Within D, some documents are relevant to the question, while the others are distractors. Explainable multi-hop QA models predict a rationale z, a minimal set of sentences across the relevant documents, in addition to predicting the answer y. We show a multi-hop QA example (with distracting documents omitted) in Figure 5.2.

x: Emily Beecham is best known for her role in a television series whose second season premiered on what date?

d_1: **[1]** *Emily Beecham is an English-American actress.* **[2]** *She is best known for her role in the AMC television series "**Into the Badlands**"* **[3]** In 2011, she received the Best Actress award at the London Independent Film Festival.

d_2: **[4]** ***Into the Badlands** is an American television series that premiered on AMC November 15, 2015* **[5]** The series features a story about a warrior and a young boy who journey through a dangerous feudal land together seeking enlightenment. **[6]** *AMC renewed the show for a 10-episode second season, which premiered on March 19, 2017.* **[7]** On April 25, 2017, AMC renewed the series for a 16-episode third season.'

$\boldsymbol{z}$: [1], [2], [4], [6]

y: March 19, 2017

Figure 5.2: A QA example. The rationale $\boldsymbol{z}$ used to derive the answer is highlighted in *blue italics*, the document-level interaction is highlighted in ***red boldface***, and the sentence-level interaction (i.e., coreference resolution) is highlighted in underline. HUG models dependencies both between documents and between sentences within a document, thus being equipped with the capacity to perform multi-hop reasoning.

5.2.1 Model

We propose the following generative model for multi-hop QA. Given the question x, we first select a subset of documents $\boldsymbol{d} = \{d_1, d_2, \ldots\} \subseteq D$. Next, within each document d_i, we select a subset of sentences $\boldsymbol{z}_i$. Finally, conditioned on the union of sentence sets from each document, $\boldsymbol{z} = \cup_i \boldsymbol{z}_i$, we generate an answer y. The only assumption we make in the model is that sentence sets are selected independently among documents. Formally, we write the model as,

$$p(\boldsymbol{d}, \boldsymbol{z}, y \mid x) = \underbrace{p(\boldsymbol{d} \mid x)}_{\text{Document set selection}} \underbrace{\prod_i p(\boldsymbol{z}_i \mid d_i, x)}_{\text{Sentence set selection}} \underbrace{p(y \mid \boldsymbol{z}, x)}_{\text{Answer generation}}$$

We refer to $p(\mathbf{d}|x)$ as the document set selection model, $\prod_i p(\mathbf{z}_i|d_i, x)$ as the sentence set selection model, and $p(y|\mathbf{z}, x)$ as the answer generation model.

Document Set Selection We select a set of documents $\boldsymbol{d}$ by directly parameterizing a distribution over all valid document sets. We rely on a document set scoring function $f(\boldsymbol{d}, x)$, which captures both the relevance of the document set $\boldsymbol{d}$ to the question x, as well as the dependencies among the documents in the set. The document set selection model is given by

$$p(\boldsymbol{d} \mid x) \propto \exp(f(\boldsymbol{d}, x)).$$

This distribution is globally normalized over all valid subsets of documents D, requiring the evaluation of the document scoring function f on all valid document subsets. Document set validity is dataset specific, and is discussed in Section 5.3.

For efficiency, the document set scoring function f first computes embeddings of each document in the set $\boldsymbol{d}$ independently, then combines them with a neural network (MLP). Formally, let $\text{emb} : \mathcal{V}^* \to \mathbb{R}^n$ be an embedding function that maps a sequence of text to an n-dimension vector, where $\mathcal{V}$ is the vocabulary. Let $\text{MLP} : \mathbb{R}^{3n} \to \mathbb{R}$ be a multilayer perception:

$$\text{MLP}(x) = W_2 \text{ReLU}(W_1 x).$$

The document set scoring function is given by:

$$f(\boldsymbol{d}, x) = \text{MLP}(\text{emb}(d_1, x), \ldots).$$

For document pairs, we specialize this to

$$f(\boldsymbol{d}, x) = \text{MLP}([\text{emb}(d_1, x), \text{emb}(d_2, x), s(d_1, d_2)])$$

where

$$s(d_1, d_2) = |\text{emb}(d_1, x) - \text{emb}(d_2, x)|.$$

This can be extended to large document sets:

$$f(\boldsymbol{d}, x) = \text{MLP}\left(\sum_i \text{emb}(d_i, x)\right).$$

We provide the details of the embedding function below, as part of the sentence selection model description.

Sentence Set Selection Within each document d_i, we select $\boldsymbol{z}_i \in \mathcal{P}(d_i)$, a power set of all sentences in d_i. We rely on a sentence set scoring function $g(\boldsymbol{z}_i, x)$, similar to the document set scoring function, which captures all relationships between selected sentences and the question. The sentence set selection model is given by

$$p(\boldsymbol{z}_i \mid d_i, x) \propto \exp(g(\boldsymbol{z}_i, x)),$$

which is globally normalized over all valid subsets of sentences in the document d_i.

Computing $p(\boldsymbol{z}_i \mid d_i, x)$ requires enumerating all sentence subsets, which is intractable. We instead extend the approach of Li et al. (2022), which obtains document and contextual sentence representations in a single encoding step. We insert a special `[SPC]` token at the beginning of each sentence, shown here at positions $k_1, k_2, \ldots,$

$$\begin{array}{ccc} u = \texttt{[CLS]}\{\texttt{x}\}\texttt{[SEP]} & \texttt{[SPC]}\ldots & \texttt{[SPC]}\ldots. \\ 0 & k_1 & k_2 \end{array}$$

We then obtain sentence subset $\text{emb}(\boldsymbol{z}_i)$ embeddings by feeding this to an encoder-only model such as BERT Devlin et al. (2019) and taking the average of the contextual embeddings of the special tokens corresponding to the sentences in $\boldsymbol{z}_i$:

$$\text{emb}(\boldsymbol{z}_i, x) = \frac{1}{|\boldsymbol{z}_i|} \sum_{j \in \boldsymbol{z}_i} \text{encoder}(u)_{k_j}.$$

Finally, we score the average embedding by taking the dot product with a learnable vector v:

$$g(\boldsymbol{z}_i, x) = v^T \text{emb}(\boldsymbol{z}_i, x).$$

In practice, encoder methods have a maximum input length that prevents encoding full documents. To address this limitation, we partition documents into slices of m

sentences, where m is less than the maximum length, and compute the embedding for each slice individually. We denote a slice for a document d as $d^{i:j}$ that starts at the ith sentence and ends before the jth sentence. Let $i \in \text{range}(0, |d|, p)$, we approximate $\text{emb}(d)$ by the following aggregation,

$$\text{emb}(d) \approx \left\lfloor \frac{|d|}{m} \right\rfloor \sum_i \text{emb}(d^{i:i+p}).$$

We set m to 3 for HotpotQA, 5 for FEVER, and 9 for MultiRC.

Answer Generation Parameterization of the answer generation model, $p(y \mid \boldsymbol{z}, x)$, is done using a sequence-to-sequence model where the question and rational are fed to an encoder, and that answer is generated. This process is complicated by the fact that answers can take on different forms, depending on specific QA tasks such as Boolean QA, multiple-choice QA, extractive QA, and abstractive QA, etc. We can therefore use a sequence-to-sequence model such as BART Lewis et al. (2020a) and T5 Raffel et al. (2020), and provide prompt templates for the different variants of the task, given in the next section.

5.2.2 Training and Inference

To learn an explainable multi-hop QA system, HUG optimizes an approximation of the marginal likelihood. The marginal likelihood,

$$\mathcal{L}(\theta) = \log \sum_{\boldsymbol{d},\boldsymbol{z}} p_\theta(y, \boldsymbol{z}, \boldsymbol{d} \mid x),$$

is intractable, as it requires computing $p(y \mid \boldsymbol{z}, x)$ under the answer generation model for every valid set of sentences across documents. We instead optimize a top-K Viterbi approximation of the marginal likelihood. Given

$$\mathcal{S}^k = \arg\operatorname*{topk}_{\boldsymbol{d}} p_\theta(\boldsymbol{d} \mid x)$$
$$\mathcal{S}^k_{\boldsymbol{d}} = \arg\operatorname*{topk}_{\boldsymbol{z}} \prod_i p_\theta(\boldsymbol{z}_i \mid d_i, x),$$

we use the following approximation of the marginal likelihood as our training objective:

$$\mathcal{L}(\theta) \approx \log \sum_{\boldsymbol{d} \in \mathcal{S}^k} \sum_{\boldsymbol{z} \in \mathcal{S}^k_d} p_\theta(y, \boldsymbol{z}, \boldsymbol{d} \mid x).$$

At test time, we must choose the best documents and rationales. Similar to training, we first choose the most likely pair of documents from S^1, then the most likely rationale from $S^1_{\boldsymbol{d}}$. Finally, for span-based QA, we generate an answer by performing greedy search on the answer generation model $p(y|\boldsymbol{z}, x)$; for Boolean QA or multiple-choice QA, we normalize the answers between different choices and take $\arg\max_y p(y|\boldsymbol{z}, x)$.

5.3 Experimental Setup

Datasets and Their Representations. We evaluate HUG on four multi-hop QA datasets: HotpotQA in the distractor setting (Yang et al., 2018), MuSiQue with answerable questions (Trivedi et al., 2022), FEVER (Thorne et al., 2018), and MultiRC (Khashabi et al., 2018). HotpotQA and MuSiQue are extractive QA datasets that require reasoning over multiple Wikipedia documents and identifying a span of text as the answer. In HotpotQA, each example contains ten candidate documents, and we must identify exactly two documents ($|D| = 2$) that are relevant. FEVER is a fact checking dataset that requires verifying claims made based on Wikipedia articles ($|D| = 1$). MultiRC is a multiple-choice QA dataset collected from diverse sources of documents including

BQA	In	*A claim to be verified is that* $\{x\}$ *We have following facts:* $\{\boldsymbol{z}\}$
	Out	*The claim is thus* {supported/refuted}.
	In	*A claim to be verified is that* Steve Wozniak designed homes. *We have following facts:* Steve Wozniak primarily designed the 1977 Apple II , known as one of the first highly successful mass-produced microcomputers.
	Out	*The claim is thus* refuted.
MCQ	In	*Question:* $\{x\}$ `[SEP]` $\{\boldsymbol{z}\}$
	Out	*Answer:* $\{y_1\}$ ({correct/wrong}) `[SEP]` *Answer:* $\{y_2\}$ ({correct/wrong}) `[SEP]` ...
	In	*Question:* Name few objects said to be in or on Allan 's desk. `[SEP]` Opening a side drawer, he took out a piece of paper and his inkpot.
	Out	*Answer:* Eraser (wrong) `[SEP]` *Answer:* Inkpot (correct) `[SEP]` *Answer:* Pen (correct)
EQA	In	$\{x\}$ `[SEP]` $\{\boldsymbol{z}\}$
	Out	$\{y\}$
	In	Which American railroad, located in Southwestern Montana and Idaho, was backed by the Northern Pacific Railway? `[SEP]` The Gilmore and Pittsburgh Railroad (G&P), now defunct, was an American railroad located in southwestern Montana and east-central Idaho.
	Out	Gilmore and Pittsburgh Railroad

Table 5.1: Seq-to-Seq Prompt templates for QA examples for FEVER as Boolean QA (BQA), MultiRC as multiple-choice QA (MCQ), and HotpotQA as extractive QA (EQA). Templates are in purple cells, followed by specific examples in green cells. Template keywords are highlighted in *red italics*.

narrative stories and news articles, and their questions can only be answered by reasoning over multiple sentences ($|D| = 1$). Unlike conventional multi-choice tasks (Lai et al., 2017, Richardson et al., 2013), MultiRC does not pre-specify the number of correct answer choices, resulting in a more challenging setting. For FEVER and MultiRC, we consider the ERASER version (DeYoung et al., 2020), where rationale annotations are made cleaner and evaluation metrics are provided.

In Table 5.1, we demonstrate how to convert QA examples of these three datasets to a natural language prompt format (Brown et al., 2020). In FEVER, a claim x needs to be classified as whether it is supported (1) or refuted (0) given the accompanying documents D. For MultiRC, there is a varying number of choices per example, and an unknown number of the choices is correct; we stack all answer choices attached with their truth values as outputs to supervise the model. Finally, extractive QA can naturally be formulated as a text-to-text problem, where the input is x [SEP] $\boldsymbol{z}$, and the output is y.

METRICS AND COMPARISON SYSTEMS. We compute F1 scores for rationale and document selection, and answer prediction. F1 scores for rationales are computed at the sentence level. Because the three QA datasets are in different formats, F1 scores for answers are computed differently. For extractive QA, F1 scores are computed at the token level for the answer spans. For Boolean QA and multiple-choice QA, F1 scores measure categorical answers.

For each dataset, we compare to (1) state-of-the-art approaches that require no rationale supervision and (2) at least one fully supervised method (i.e., answers and rationales available for training). The latter provides an upper bound on performance.
– **On HotpotQA and MuSiQue**, we compare to a rule-based approaches, BM25, and RAG (Lewis et al., 2020b) as unsupervised baselines. We note that RAG only performs document-level retrieval, and therefore its current form cannot be directly applied to identifying sentence-level rationales. We modify RAG to treat a sentence as a document, and at inference we take the top-3 sentences to be the rationale as it results in the highest sentence F1 scores. For fair comparison, we parameterize RAG in the same way as we parameterize HUG. We also consider a semi-supervised approach – CHAIN (Chen et al., 2019); they assume no access to gold rationale annotations and supervise their model on silver rationales produced with external entity taggers. The fully supervised system we consider is SAE (Tu et al., 2020). Both CHAIN and SAE use

RoBERTa-large as sentence encoders.

– On FEVER and MultiRC, we also compare to RAG as an unsupervised baseline (predicting top-2 sentences for MultiRC, and top-1 sentence for FEVER). Additionally, we consider diagnostics-guided explanation generation (Diagnostics) (Atanasova et al., 2022) and faithful rationales (Faithful) (Glockner et al., 2020). Both of these methods have two variants – one trained with rationale supervision (denoted by $\mathbb{RS}$-*) and the other trained without rationale annotations (denoted by $\mathbb{RU}$-*). On MultiRC, We also compare to WT5 (Narang et al., 2020).

Implementation and Hyperparameters. We test HUG with language models of both small and large sizes. For the small version (HUG-Small), we use distilBERT (Sanh et al., 2019) as the encoder and BART-base as the seq2seq model. For the large version (HUG), we use RoBERTa-large as the encoder and BART-large as the seq2seq model. We only test RAG in the small version.

We implement HUG with Hugging Face Transformers (Wolf et al., 2020). We perform grid search with learning rates {5e-6, 1e-5, 2e-5} and batch sizes {2,4,8,16} for both HUG and RAG. We train our system for 3 epochs for HotpotQA and 5 epochs for the other three datasets. We warm up the learning rate with first 10% examples. We choose the checkpoint that has the highest answer F1 score on the validation set. We consider rationales up to four sentences for HotpotQA and rationales up to three sentences for the other datasets. Finally, we take S^{10} and $S_{\boldsymbol{d}}^{9}$ for HotpotQA and $S_{\boldsymbol{d}}^{80}$ for MultiRC. Because we remove rationales whose sentences are not contiguous in FEVER, we are able to compute the exact likelihood without top-k sampling.

		HotpotQA		MuSiQue	
	#Params	**Sent F1**	Ans F1	**Sent F1**	Ans F1
BM25	-	40.5	-	12.9	-
RAG-Small	221M	49.0	62.8	32.0	24.2
HUG-Small	221M	67.1	66.8	34.2	25.1
HUG	761M	**72.5**	73.5	**44.4**	39.1
Chain (semi-supervised)	355M	64.5	66.0	-	-
SAE (supervised)	790M	87.4	80.8	75.2	52.3

Table 5.2: Performance comparison on predicting rationales and answers on HotpotQA and MuSiQue.

		FEVER		MultiRC	
	#Params	**Sent F1**	Ans F1	**Sent F1**	Ans F1
ℝ𝕌-Faithful	110M	83.8	90.6	27.5	67.7
ℝ𝕌-Diagnostics	110M	56.1	-	38.1	-
RAG-Small	221M	80.7	92.0	44.9	70.0
HUG-Small	221M	81.5	91.6	48.4	74.3
HUG	761M	**84.1**	94.3	**55.6**	75.5
ℝ𝕊-Faithful (supervised)	110M	91.4	91.1	46.1	67.4
ℝ𝕊-Diagnostics (supervised)	110M	94.4	89.7	79.4	71.7

Table 5.3: Performance comparison on predicting rationales and answers on Eraser-FEVER and Eraser-MultiRC. HUG has more parameters due to its use of a seq2seq model for answer generation.

5.4 Results

HotpotQA. We summarize the results on HotpotQA in Table 5.2. HUG-Small outperforms the best unsupervised approach RAG-Small by 18 sentence F1 points, demonstrating superior multi-hop reasoning abilities. HUG-Small, despite having fewer parameters, outperforms the semi-supervised Chain on predicting rationales and is comparable to Chain on predicting answers. While Chain explicitly exploits the heuristics used in the data collection process for HotpotQA, HUG-Small is able to learn such heuristics in a fully automatic way. Finally, the gap between HUG and SAE, a fully supervised method that is given both rationale and answer annotations, remains large.

MuSiQue. Table 5.2 shows that HUG-Small is better at both identifying rationales and predicting answers than RAG-Small, the best-performing unsupervised baseline. Due to the difficulty of MuSiQue, it is harder for an unsupervised method to learn to select rationales – the gap between the supervised method and the best unsupervised method on this dataset is greater than the gap on HotpotQA.

FEVER We summarize the results on FEVER in Table 5.3. In terms of selecting rationales in an unsupervised manner, HUG-Small outperforms $\mathbb{RU}$-Diagnostics, performs similarly to RAG-Small, and underperforms $\mathbb{RU}$-Faithful by a small margin. Because FEVER mostly only requires single-hop reasoning, HUG-Small does not improve over the previous methods. On predicting answers, HUG-Small outperforms $\mathbb{RU}$-Faithful but underperforms RAG-Small. Compared to the supervised versions of $\mathbb{RS}$-Diagnostics and $\mathbb{RS}$-Faithful, which have access to both the answers and rationales during training, the gap between HUG-Small's and their rationale scores (sentence F1) understandably remains large.

MultiRC Table 5.3 shows that HUG-Small outperforms all comparison models including RAG-Small – the best competing approach without rationale supervision – by 3.5 sentence F1 points and 4.3 answer F1 points.

		Sent F1	Doc F1	Ans F1
Comparison	HUG-Ind	**78.9**	**92.9**	64.8
	HUG	78.1	91.1	**69.7**
Bridge	HUG-Ind	55.2	68.6	71.6
	HUG	**71.0**	**87.3**	**75.7**
Combined	HUG-Ind	60.0	73.4	69.1
	HUG	**72.5**	**88.0**	**73.5**

Table 5.4: Rationale selection performance broken down by different types of reasoning.

Scaling HUG to Larger Models. On all three datasets, by increasing the number of model parameters, HUG can consistently achieve better performance. Additionally, as the number of reasoning hops increases, HUG can more benefit from the larger language models – compared to HUG-Small, HUG has the least improvement on FEVER and had the most improvement on MuSiQue.

5.5 Analysis

Document Dependencies HUG explicitly models the dependencies between documents for multihop reasoning. We consider independent document selection [1] to see whether modeling document dependency is necessary for HotpotQA.

To understand how document modeling impacts rationale selection performance, we break the performance down by the reasoning types proposed in Yang et al. (2018):

[1] For independent document selection, we train a different document selection model that factors as

$$p(\boldsymbol{d} \mid x) = \prod_{d \in \boldsymbol{d}} p(d \mid x).$$

Exact marginalization of $\boldsymbol{d}, \boldsymbol{z}$ is still intractable. We thus only marginalize over $\mathcal{S}^5$ and $\mathcal{S}^5_{\boldsymbol{d}}$. At inference, we choose

$$\boldsymbol{d} = \arg \underset{d}{\text{topk}}\, p(d \mid x),$$

where k is the number of documents pre-specified by the task.

comparison-based reasoning and bridge-based reasoning. In comparison-based reasoning, relevant documents independently contribute to the answer, whereas for bridge-based reasoning, relevant documents require connections to previously selected documents. Table 5.4 summarizes answer F1 scores, document F1 scores, and sentence F1 scores. While HUG-Ind is slightly better at comparison-based reasoning than the joint model, it fails at bridge-based reasoning; this result thus confirms the necessity of modeling the dependency between documents. We also note that HUG-Ind and HUG have similar performance in predicting answers, but the gap between how accurate they select rationales is large, suggesting that HUG-Ind often derives answers with wrong reasoning. Overall, HUG is better than HUG-Ind at both predicting answers and selecting rationales.

In addition to the quantitative analysis, we also qualitatively compare the two models in Figure 5.3. When considering paragraphs independently, documents A and C share the most entities with the question (i.e., Copsi, earl of Northumbria, and Two Rivers), so they are more likely to lead to the answer. However, the correct documents are A and B. Deriving B not only depends on the question but also further requires knowing the information from A. Therefore, while having the independent document selection model can improve efficiency because it only performs one-step reasoning, the joint document selection model is necessary when reasoning steps depend on one another.

Role of Answer Generation HUG uses a generative model (BART) to parameterize $p(y \mid \boldsymbol{z}, x)$. An alternative approach would be to use a classification model such as RoBERTa (Liu et al., 2019a) to predict answers for FEVER and MultiRC (HotpotQA requires a generative model).

Q: When **Copsi** was made **earl of Northumbria** he went to reside in a town at the confluence of which **two rivers**?

Document A, Copsi:
Copsi survived Tostig's defeat at Stamford Bridge, and when William the Conqueror prevailed at Hastings he travelled, in March 1067, to pay William homage at Barking (where William was staying while his tower was being constructed in London). *In return, William made **Copsi earl of Northumbria** and sent him back to York.*

Document B, York:
York is a historic walled city at the confluence of the rivers Ouse and Foss in North Yorkshire, England. The municipality is the traditional county town of the historic county of Yorkshire to which it gives its name.

Document C, Two Rivers Press:
Two Rivers Press is an independent publishing house, based in the English town of Reading. Two Rivers Press was founded in 1994 by Peter Hay (1951–2003).

A: Ouse and Foss

Figure 5.3: A HotpotQA example where there is a dependency between two supporting documents, and thus selecting the second document independent of the first one results in insufficient information. Correct rationale is highlighted in *blue italics*. Entity overlaps between questions and documents are in **red boldface**. HUG-Ind's predicted Documents B and C, whose reasoning remains at the surface level as they share the most entities with the question. HUG predicted Documents A and B, which demonstrates its ability of understanding dependency between documents.

Interestingly, Table 5.5 shows the choice of answer model significantly impacts the ability of HUG to learn a rationale model. On FEVER, where claims cannot be verified without the corresponding rationales, BART and RoBERTa perform similarly. However, on MultiRC, where questions can often be answered without information in accompanying documents, the best Generative model outperforms the best Classification model by over 32 sentence F1 points.

Figure 5.4 shows an example of such a question where the answers can be guessed by the classification model using commonsense knowledge to reason about law. Gen-

Answer Model	Pred Type	FEVER	MultiRC
		Sent F1	
BART	Generate	81.5	48.4
RoBERTa	Classify	82.0	20.0

Table 5.5: Comparison on sentence F1 scores between different parameterization choices of $P(y \mid \boldsymbol{z}, x)$.

Q: What did the judge tell Mr. Thorndike about the law?
A1: *Cannot be swayed by wealth or political influences.*
A2: *The law is not vindictive.*
A3: *It was not vindictive.*
A4: It was unjust.
A5: It was vindictive.
A6: *The judge told Mr. Thorndike that the law is not vindictive. He said the law only wishes to be just. Judge said the law cannot be swayed by wealth or political influences.*

Figure 5.4: A test example from MultiRC that can be answered with commonsense reasoning and thus requires no accompanying documents. Correct answers are highlighted in *blue italics*.

erative models need to assign a high probability to every token in the answer, and we hypothesize that they make better use of the answer supervision.

	HUG	FAITHFUL	Ratio
Training	11,544.2	444.0	26
Inference	39.0	3.9	10

Table 5.6: Runtime comparison (in seconds). HUG uses 80 rationale samples at training time, and the argmax rationale at inference.

SPEED EVALUATION. While HUG obtains strong sentence F1 scores, training is more expensive because the model must consider a set of rationales for every example. In particular, the answer model $p(y \mid \boldsymbol{z}, x)$ must be run for *every* sampled $\boldsymbol{z}$ for each training example. At inference, the answer model requires only a single evaluation of $p(y \mid \boldsymbol{z}, x)$ for $\arg\max_{\boldsymbol{z}} p(\boldsymbol{z} \mid x)$. We empirically measure the runtime overhead of HUG compared to FAITHFUL on MultiRC, using 80 samples of $\boldsymbol{z}$ at training time. We report the total training time and inference time in Table 5.6. Compared to FAITHFUL, HUG takes longer to train and to predict.

5.6 Related Work

Explainable methods for multi-hop QA. Active research has been devoted to collecting human rationales for a wide range of QA tasks; a recent survey has identified 65 datasets that provide explanation annotations Wiegreffe & Marasovic (2021). The appearance of such datasets has enabled rapid progress in supervised methods for extracting reasoning chains; we refer readers to Thayaparan et al. (2020) for a comprehensive survey. While these supervised methods such as Qi et al. (2019) have achieved tremendous success on retrieving rationales, even in the open-domain setting [2], they can only be applied when rationale annotations are available. However, such annotations do not always exist – Welbl et al. (2018) and Yu et al. (2022) propose two complex reasoning datasets that do not have rationale annotations. In these cases, we need unsupervised rationale selection methods to still build explainable QA systems.

Other works have explored multi-hop QA with only answer supervision but not rationale supervision. As in our work, Retrieve and Generate (RAG) Lewis et al. (2020b), treats the rationale as a latent variable; however, in RAG the open retrieval-stage is to find a single document, ignoring the connections between different docu-

[2] While we only consider the distractor QA setting in this work, HUG can be combined with a rule-based retrieval system such as BM25 to be adapted to the open-domain setting.

ments. RAG also does not produce sentence-level rationales. Other works consider sentence rationales: Glockner et al. (2020) compute a score for every sentence pair and pick the sentence pair that has the highest score as the rationale for answer prediction, and Atanasova et al. (2022) perform binary classification on whether individual sentences are included in the rationale with added constraints such as consistency and faithfulness. The shared limitation of these methods is that they do not capture the dependency between more than two pieces of information. HUG overcomes this limitation by performing multi-hop reasoning as document set prediction and sentence set prediction.

Outside of unsupervised methods, Chen et al. (2019) propose a semi-supervised method which collects silver rationale annotations. However, their method is limited to bridge-based questions, which is only one form of multi-hop reasoning; the other types can be found in Trivedi et al. (2022).

Rationales as latent variables. A focus for rationale methods in NLP outside of multi-hop QA has been identifying subsets of input tokens to justify decisions. For text classification, Lei et al. (2016), Bastings et al. (2019), and Chen & Ji (2020) frame rationales as minimal subsets of input tokens. For multi-hop QA, where input tokens are too granular a representation for rationales, treating sentences as rationales within long documents leads to the challenges of hierarchical selection and the representation of long documents; both of which we address with HUG.

Outside of using input tokens for rationales, Zhou et al. (2020) assume rationales take the form of unconstrained text. While flexible, this approach leads to computationally expensive training methods. Therefore, we constrain rationales to be a set of sentences from given documents, which both accommodates the production of useful intermediate reasoning steps and keeps training tractable.

Unsupervised retrieval. A task closely related to our setting (i.e., no access to rationale supervision) is unsupervised retrieval, which searches for sentences relevant to the questions but does not predict answers. For example, one could apply Yadav et al. (2019), Yadav et al. (2020), Zhao et al. (2021) and Xu et al. (2021) to first identify the rationale for a multi-hop QA example and predict an answer based only on the rationale rather than on the entire document, so that the answer prediction is more constrained. However, HUG may be preferred over these unsupervised retrieval methods because they assume specific types of QA formats Xu et al. (2021), but HUG works on any type of QA problems, and 2) while other works also propose to model rationales as a latent variable, we additionally introduce the hierarchical structure in our probabilistic model, enabling efficient inference.

5.7 Conclusion

In this chapter, we presented HUG, a method for question answering with structured state tracking representations, where the state representations explicitly selected sets of supporting evidence. Experimental results demonstrate that HUG outperforms other state-of-the-art methods that do not rely on rationale labels. An important future direction would be to scale this process up, allowing both retrievers and generators to reason over the state of the evidence gathered so far.

This chapter concludes our study of question answering. In the next chapter, we will study our final application of state tracking: dialogue.

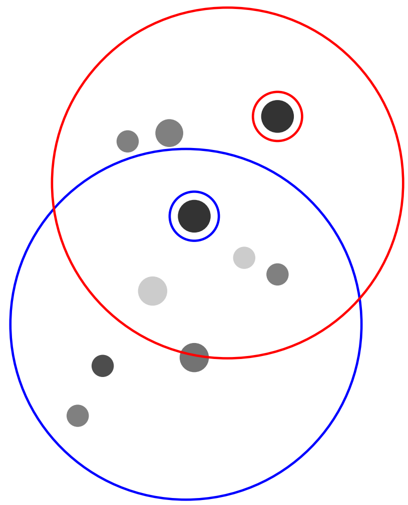
6

Modeling Perspective-Dependent Ambiguity in Collaborative Dialogue

6.1 Introduction

The final state tracking application we study is dialogue, where the state representation must summarize the information exchanged so far. This chapter proposes a state tracking representation that explicitly captures ambiguity in a visually grounded dialogue.

Ambiguity is a key impediment to collaborative dialogue that makes linguistics actions, such as reference resolution and generation, difficult to execute. One cause of ambiguity in rounded settings is differences in perspectives (Brown-Schmidt &



A: I have one large black dot by itself. Do you have it?
P: Yes, I do have that.
A: Let's pick that one.
P: <select> red
A: <select> blue

Figure 6.1: An example of perspective-dependent ambiguity from a dialogue agent and human partner playing OneCommon, taken from the evaluation of Fried et al. (2021). The players have different but overlapping circular views of a shared board, which contains dots of different shades, shapes, and sizes. The agent and partner must collaborate through dialogue in order to find and select a shared dot. This dialogue ends in failure, as the agent and partner did not account for perspective-ambiguity and prematurely selected different dots.

Hanna, 2011). Agents that incorrectly assume their partner shares their perspective may make errors in reference generation and resolution. We refer to ambiguity induced by different perspectives as *perspective-dependent ambiguity*.

Figure 6.1 provides an example from OneCommon, a collaborative reference game with a high degree of perspective-dependent ambiguity (Udagawa & Aizawa, 2019). An agent and partner are given different but overlapping views of a game board and must jointly identify a shared dot through dialogue. Perspective dependent ambiguity stems from the different but overlapping views, and the degree of ambiguity depends on the number of overlapping dots. In this example, the agent's description (*one large black dot*) is specific in their view, but actually resolves to a different dot in their partner's view — unknown to both the agent and partner. The incorrect resolution and confirmation have an immediate irrevocable consequence: The agent and partner select different dots, failing the game.

This false positive confirmation results from an *egocentric* heuristic (Keysar et al., 2000). Agents utilizing an egocentric heuristic resolve referring expressions based mainly on their own perspective, failing to account for the fact that the perspectives have partial overlap and there are unshared items which cannot be observed by the agent. While cognitively cheap, the egocentric heuristic can lead to systemic errors. Such errors are more pronounced with larger differences in perspectives, resulting in larger degrees of perspective-dependent ambiguity. Errors of this form can result in irrevocable mistakes that do not get corrected in follow-up interactions (Keysar et al., 2000).

In this work, we propose a dialogue planner that infers the partner's perspective through symbolic state tracking. Our approach centers on a partner model that predicts the partner's response given a question from the agent and the partner's unobserved perspective, modeled in our probabilistic state tracking framework. The partner model reasons explicitly over shared and unshared aspects of the partner perspective, marginalizing over the unshared aspects. The agent uses the partner model to plan symbolically, asking maximally informative questions by maximizing the expected information gain with respect to the latent partner perspective. After observing the partner's response to the utterance, the agent uses the partner model to conservatively update its belief over shared perspective.

Experiments show that planning through the partner model outperforms a state-of-the-art fully supervised dialogue agent (Fried et al., 2021). The supervised approach chooses plans based on an egocentric heuristic, and is limited by the strategies demonstrated in its training data. When applied to OneCommon, our approach utilizes richer plans than those demonstrated by humans and achieves a symbolic[1] selfplay success rate of 85.2%, a large improvement over the prior state-of-the-art language selfplay success rate of 62.4% (Fried et al., 2021).

6.2 Related work

Perspective in common ground The study of perspective and its influence on *common ground*, the shared knowledge between dialogue agents, has its roots in pragmatics and linguistics (Stalnaker, 1970, Clark & Marshall, 1981, Clark et al., 1983). Behavioural experiments studying the egocentric heuristic found that it can lead to irrevocable errors in human studies (Keysar et al., 2000). Computational work by Liu et al. (2021) and Doğan et al. (2020) utilize perspective to generate and understand referring expressions in a fully shared environment, but from different locations. In contrast to these works, where the perspective of the partner is known and conditioned on, in our setting the partner perspective is inferred and the environment is partially shared between partners.

Perspective captures more than just spatial properties. Partner capability was considered in the context of reference games (Corona Rodriguez et al., 2019), where partner capabilities such as color-blindness were inferred in a single-turn visual reference game. While their approach did not model partner perspectives, we propose to directly model the partner perspectives as a latent variable.

[1] Symbolic selfplay for OneCommon allows agents to directly communicate the symbolic representation of plans instead of describing them in natural language.

Informative question generation Work in question generation formulates information-seeking conversation as a partially observable Markov decision process. There is one information seeker and one information holder. Methods use a single-turn partner-model heuristic to generate questions (Rao & III, 2018, Yu et al., 2019). In this setting, the unobserved target referent is the only source of partial observability, meaning that information exchanged is reliable up to errors in language. In contrast, the different perspectives in OneCommon and resulting perspective-dependent ambiguity have a large effect on communication, often turning unambiguous utterances in the agent's perspective into ambiguous ones for the partner.

Task-oriented dialogue Task-oriented dialogue often uses multi-turn model-based planning. Here planning is performed in the space of language via rollouts or tree-search (Lewis et al., 2017, Yarats & Lewis, 2017, Shridhar & Hsu, 2018, Jang et al., 2020). While we use a single-turn planning heuristic, our focus on improving partner models is complementary to multi-turn planning and can be combined in future work.

Reference generation There is a long line of work on referring expression generation (Heeman & Hirst, 1995, Dale & Reiter, 1995, Jordan & Walker, 2005, Krahmer & van Deemter, 2012, Takmaz et al., 2020, *inter alia*), particularly in grounded settings (Dale & Viethen, 2009, Mao et al., 2016, Yu et al., 2017, Takmaz et al., 2020, *inter alia*). The OneCommon (Udagawa & Aizawa, 2019) dialogue setting poses challenges for generation due to its partial observability and the non-identical views of its participants. Our method is most closely related to the state-of-the-art approach to OneCommon (Fried et al., 2021) which forms the backbone of our method. We focus on relaxing the egocentric assumptions of this method by incorporating symbolic information-seeking planning and generation.

6.3 Partner Modeling in Reference Games

Collaborative reference games pair an agent and a partner in order to agree on a shared object through natural language dialogue. At each turn, the agent or partner may decide to terminate the game and make a selection. Once either the agent or their partner terminates, the other player must also act (without observing the other's choice). If both agent and partner agree, both win; otherwise, both fail.

Our approach to reference games separates planning of utterances (choosing what to talk about) from surface realization (choosing how to say it). At each turn, our agent produces an utterance plan x by using a *partner model*, which simulates the partner's possible responses, y, given their hidden perspective, z. The agent uses the partner model to infer the partner's perspective and predict the partner's responses to the agent's plans. We first give an overview of the partner model and planning procedure in this section.

Partner model We model the partner's perspective as a latent variable, infer the value for this variable over the course of a game, and use it to plan generation. This contrasts with a typical egocentric heuristic, as used in Fried et al. (2021), which assumes the partner's perspective is identical to the agent's.

The partner model predicts a distribution over the partner response y given the agent plan x under the latent shared perspective z, and decomposes as:

$$p(y \mid x) = \sum_z p(y \mid x, z) p(z).$$

Planning The agent uses the partner model to plan what to say next, by choosing the plan x that maximizes the expected information gain (Lindley, 1956) about the shared perspective z, defined as

$$\underset{x}{\operatorname{argmax}}\, H[z] - \mathbb{E}_{y|x}\left[H[z \mid x, y]\right],$$

where $H[z]$ is the entropy of the prior[2] and $H[z \mid x, y]$ the posterior, which requires marginalizing over z.

Belief update After observing the partner response y, the agent updates its belief $p(z)$ over the shared with Bayes' rule:

$$p(z \mid x, y) = p(y \mid x, z)p(z) / \sum_{z} p(y, z \mid x)$$

This is performed iteratively after each turn, and requires marginalizing over possible shared perspectives.

Selection After gathering information through planning and incorporating information through belief updates, the agent must decide when it has built enough common ground, collaboratively identifying a shared dot with its partner. We set a threshold on the belief entropy, $H[z]$, which determines when the agent should transition from information gathering to ending the game.

6.4 Planning in OneCommon

We focus on applying partner modeling to OneCommon (Udagawa & Aizawa, 2019), which represents a class of collaborative reference games (He et al., 2017, Haber et al.,

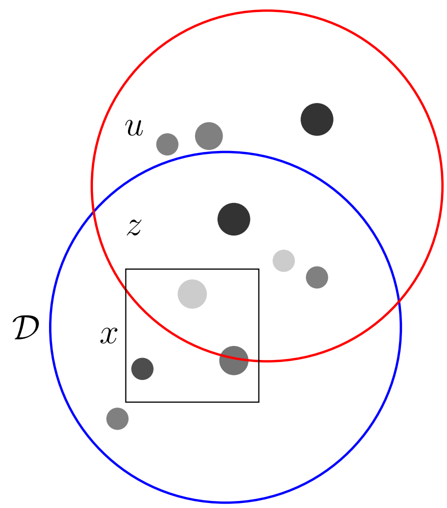
[2]The prior entropy $H[z]$ in the definition of information gain is constant with respect to the plan x, and can be dropped from the objective.

2019) where only a subset of each player's perspective is shared, resulting in perspective-dependent ambiguity.

In OneCommon, the agent's known perspective $\mathcal{D}$ consists of 7 dots in its view. Each dot has a set of features: size, color, and position in the 2D plane. All features are continuous. The main challenge of the game is that the partner perspective is also a view of 7 dots, Between 4–6 of those dots are shared with the agent perspective which we denote as the shared perspective z. Additionally there are a set of unshared dots u the fill out the partner perspective. An example is given in Figure 6.2. Note that a smaller number of shared dots increases the likelihood that plans get misresolved to unshared dots, increasing perspective-dependent ambiguity.

The agent communicates with the partner by producing an utterance plan, x, which it then describes in natural language. This plan is a subset of the dots in the agent view, $x \subseteq \mathcal{D}$, that the agent will ask the partner about. The partner gives a response y to the plan x, given their perspective z. In OneCommon, the partner responds in natural language; however, the partner model only models the partner response as a confirmation $y \in \{\textsc{yes}, \textsc{no}\}$, obtained by classifying natural language responses.

Exact planning is intractable because the objects in the partner perspective have continuous-valued features. In this section, we describe simplifying assumptions for the partner model and inference procedure that make planning tractable.



Do you have a triangle of one gray dot ...

$y = $ No

Figure 6.2: In OneCommon, the agent's perspective $\mathcal{D}$ is represented by the large blue circle, and the partner's unobserved perspective by the red. The shared dots z are in both perspectives, while the unshared dots u are only in the red circle. The agent plan x is given by the dots in the box, and also described in language. The partner response y is a binary confirmation.

6.4.1 Partner model

We build a partner model by factoring the shared perspective z and partner response y as illustrated in Figure 6.2. Formally,

$$\begin{aligned} p(y \mid x) &= \sum_{z} p(y \mid x, z) p(z) \\ &= \sum_{z,u} p(y \mid x, z, u) p(z) p(u), \end{aligned}$$

where we introduce the latent variable u representing the unshared dots in the partner perspective.

The shared dot prior, $p(z)$, is a distribution over subsets of $\mathcal{D}$, indicating which dots in the agent perspective $\mathcal{D}$ are shared with the partner. The model $p(z)$ is initially uniform over dot subsets at the start of a game, but is updated given evidence from the partner response y at the end of each turn, $p(z \mid x, y)$. For notational simplicity we focus on the first turn.

The unshared dot prior, $p(u)$, is a distribution over the remaining partner dots. Since the dots in u are unobserved by the agent, we parameterize $p(u \mid s)$ using a uniform distribution over discretized features for each dot. We ignore spatial features for dots in u and discretize the other originally continuous features: size and color.[3]

The confirmation model, $p(y \mid x, z)$, checks whether a partner will confirm or deny the agent plan. The partner confirms if they are able to resolve the plan x to their perspective. Given a fully observed z and u, resolution of a plan x is performed by matching the features of x to z and u. There are no trained parameters in resolution, as it depends only on the features of dots in x, z, and u. For OneCommon, the set of features used for each plan x is given by the shape and size each dot in the plan. For simplicity, we bucket each feature into 3 bins based on the range of each feature. The pairwise positions, limited to above-left, above-right, below-left, and below-right, are also contained in the feature set. We provide an example of feature-based resolution in Figure 6.3.

Given a plan x, feature-based resolution must compare all the features of the plan, of which there are $O(|x|^2)$, to all partial permutations of subsets size $|x|$ taken from $\mathcal{D}$, of which there are $O(|\mathcal{D}|^{|x|})$. This can be precomputed at the start of a dialogue.

In order to avoid jointly enumerating z and u, the model reasons separately about z and u by making the simplifying assumption that plans are fully in z or u. This means

[3] We discretize size and color uniformly into 3 buckets based on their absolute range across OneCommon.

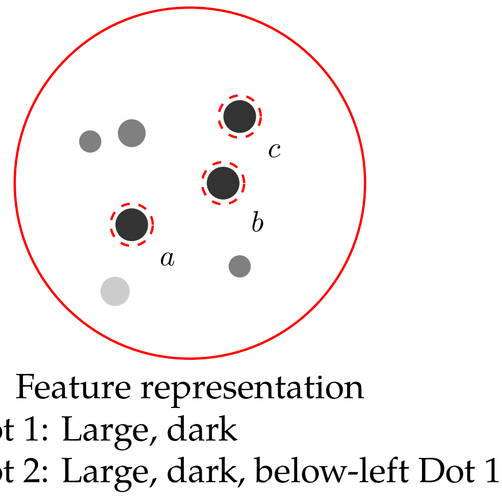


Figure 6.3: An example of feature-based resolution. The above feature representation for a pair of dots resolves to dot configurations $\{(a, b), (a, c), (b, c)\}$.

that the model will deny if part of x is in z, while the remainder is in u (and x is not fully contained in either z or u):

$$\begin{aligned} p(y = \textsc{no} \mid x) &= \sum_{z,u} p(y = \textsc{no} \mid x, z, u) p(z, u) \\ &= \sum_{z} p(y = \textsc{no} \mid x, z) p(z) \\ &\quad \cdot \sum_{u} p(y = \textsc{no} \mid x, u) p(u). \end{aligned}$$

Given the unsuccessful resolution of x to both z and u, the partner denies accurately with probability θ, a hyperparameter.

6.4.2 Inference

During inference, we need to compute $p(y \mid x)$ for all plans x, which can be done in two steps: First, we marginalize over the unshared dots u. Second, we marginalize over the possible set of shared dots z. The computational cost of marginalization is the size of the power set of $\mathcal{D}$, $O(2^{|\mathcal{D}|})$.

The marginalization over unshared dots can be computed tractably. The partner model, with the assumption that x cannot be split between z and u, is given by

$$\begin{aligned}
&p(y = \textsc{no} \mid x) \\
&= \sum_{z,u} p(y = \textsc{no} \mid x, z, u)p(z)p(u) \\
&\approx \sum_{z,u} p(y = \textsc{no} \mid x, z)p(z)p(y = \textsc{no} \mid x, u)p(u) \\
&= \sum_{z} p(y = \textsc{no} \mid x, z)p(z) \sum_{u} p(y = \textsc{no} \mid x, u)p(u) \\
&= \sum_{z} p(y = \textsc{no} \mid x, z)p(z) \sum_{u} p(y = \textsc{no}, u \mid x).
\end{aligned}$$

The probability a plan x resolves to the unshared dots u is

$$\sum_{u} p(y = \textsc{no}, u \mid x) = 1 - \theta \binom{|u|}{|x|} \frac{B^{2\cdot(|u|-|x|)}}{B^{2\cdot|x|}},$$

where B is the feature bucket size, given $|u| \geq |x|$. This relies on the assumption that spatial features are ignored when resolving to unshared dots.

We utilize $p(y|x)$, which predicts the partner's response to our plan, to compute the posterior on the shared perspective z,

$$p(z \mid x, y) = \frac{p(z, y \mid x)}{p(y \mid x)}.$$

This posterior then allows us to perform optimization over plans with respect to the expected information gain, as well as update our beliefs given the partner response.

PLANNING Planning optimizes the expected information gain with respect to the shared perspective z:

$$\operatorname*{argmin}_x \mathbb{E}_{y|x}\left[H(z \mid x, y)\right].$$

Computing $p(y \mid x)$ has cost $O(2^{|\mathcal{D}|})$, while there are also $O(2^{|\mathcal{D}|})$ plans.[4] As a result, optimizing this objective takes $O(2^{2|\mathcal{D}|})$ computation, and is performed in less than one second on CPU.

Belief update The belief update directly uses the posterior distribution $p(z \mid x, y)$, as described in Section 6.3.

During gameplay in ONECOMMON, the agent either directly observes the symbolic response y or receives a description of y in natural language. In order to process the natural language dialogue, we use a classifier to extract y from natural language. Additionally, the partner can mention dots of their own, either symbolically or described in text. The agent incorporates partner mentions into its belief by treating them as a confirmed plan. We use another classifier to extract partner mentions from text. We give the details of both the response and mention classifiers in Section 6.5.

Selection To determine when to select a dot, the agent uses a threshold on the entropy $H[z]$, given by the hyperparameter τ. The agent them communicates which dot to select by describing the configuration of four dots with the highest marginal probability of being shared, as well as the dot within that configuration that is most likely to be shared. The agent then selects the described dot.

[4]Plans x are subsets of $\mathcal{D}$ that the agent would like to ask the partner about.

6.5 Experimental setup

We evaluate our method, the Partner Planner, on the OneCommon dataset (Udagawa & Aizawa, 2019). We perform two evaluations: First, we evaluate the number of incorrectly resolved plans generated by the Partner Planner given a static, natural language dataset. Second, we evaluate the Partner Planner in dynamic, symbolic selfplay.

Static plan evaluation In order to show that resolving perspective-dependent ambiguity reduces errors, we perform automatic evaluation of plans by evaluating whether the agent plan is incorrectly resolved by the partner model. A plan is incorrectly resolved if the plan is not empty and the partner does not resolve the plan to any of the agent's intended referents. We evaluate this without language by directly feeding the feature representation of plans from the agent to the partner, who is a Partner Planner.

We generate plans given natural language dialogue history from a validation split of OneCommon following prior work (Fried et al., 2021).[5] For each turn in the human-generated dialogue, we generate a plan from our model and label that plan as either a success or failure using the procedure above. We evaluate on 518 validation games, which have different numbers of shared dots: either 4, 5, or 6. Fewer shared dots results in more perspective-dependent ambiguity.

Symbolic selfplay We also evaluate the Partner Planner on symbolic selfplay, where it plays the full OneCommon game with a copy of itself using symbolic communication. We evaluate only on the setting with the most perspective-dependent ambiguity,

[5] Prior work used 10-fold cross-validation. We use models from prior work trained on one fold, and evaluate our approach on the validation set for that fold. In particular, we use fold 1 from prior work.

4 shared dots. In symbolic selfplay, agents must perform planning, belief updates, and selection. The Partner Planner is able to exactly communicate confirmations and dot features (bucketed size, color, and relative positions). We compare the game success rate of the Partner Planner to success rate of the baseline by Fried et al. (2021) and human performance on language selfplay. Language inherently has more noise than symbolic representations, meaning symbolic selfplay is an upper bound on performance with language.

Systems We focus evaluation on the Partner Planner, which reasons about shared and unshared dots. We consider an ablated version, which does not model unshared dots.

As a baseline, we compare to an agent from prior work, which does not account for perspective-dependent ambiguity (Fried et al., 2021). The baseline model chooses plans based on the round-trip probability from an utterance back to the plan, which is an egocentric heuristic and also does not account for uncertainty over shared dots. The baseline agent chooses 8 plans and 64 utterances for each plan, then chooses the plan and utterance pair with the highest probability of recovering the plan give the utterance using a reference resolution model applied only to the agent's own context.

Hyperparameters For the Partner Planner, we set the response faithfulness probability $\theta = 0.95$. We determine the selection entropy threshold by running grid search over $\tau \in \{1, 1.5, 2\}$ on symbolic selfplay, and pick the value with the highest success rate. We parameterize the belief prior, $p(z)$, over which dots are likely to be shared with a globally normalized Ising model

$$p(z) \propto \exp(f(z)), \tag{6.1}$$

Agent	# shared 4	5	6	Total
Partner Planner	26	12	5	43 (12)
-unshared	70	61	22	153 (22)
Fried et al. (2021)	28	17	4	49 (17)

Table 6.1: The number of incorrectly resolved plans in automatic feature-based evaluation on 519 static validation dialogues (2,341 turns). The total number of errors made in the first turn of a dialogue is shown in parentheses for each agent. A lower number of incorrect plans is better.

where $f(z)$ is determined by the sum of the pairwise distances between dots in z.

The partner response classifier is a RoBERTa model (Liu et al., 2019b), with a second stage of fine-tuning performed on 147 annotated dialogue turns. We annotate the text each turn as a confirmation, denial, or no response. The model is originally fine-tuned on sentiment (Heitmann et al., 2020).

For mention classification from text, we use a mention prediction network from past work (Fried et al., 2021). The mention prediction network explicitly models relationships between mentions using a conditional random field with neural potentials.

6.6 Results

Static plan evaluation Directly modeling and marginalizing over unobserved partner perspective results in fewer errors than the egocentric heuristic from Fried et al. (2021), obtaining a 12% reduction in errors as shown in Table 6.1. Additionally, the number of incorrectly resolved plans from Planner is much lower than an ablated planner that does not model unshared dots, a 72% reduction.

We hypothesize that the baseline model of Fried et al. (2021) makes fewer errors because it is able to repeat plans mentioned in the static dialogue. The Partner Plan-

Agent	Success	Avg # turns
Partner Planner	85.2%	10.10
-unshared	77.0%	11.69
Fried et al. (2021)	62.4%	-
Human	65.8%	4.97

Table 6.2: The success rate of different agents in selfplay on the hardest setting of OneCommon, with 4 shared dots. The Partner Planner and ablated version communicate symbolically, while the Fried et al. (2021) baseline and human performance use language. A higher success rate is better. The human performance is from the OneCommon dataset (Udagawa & Aizawa, 2019).

ner does not often repeat plans, as there is little information gained. When restricted to the plan proposals from the first turn of a static human-demonstrated dialogue, where no plans can be copied, the Partner Planner outperforms the baseline by a larger margin, as shown in Table 6.1.

We note that the absolute number of resolution errors is small relative to the total number of turns. We hypothesize that this is because of the nature of static evaluation. Static evaluation considers next step plan proposals given human dialogue, preventing agents from steering the dialogue themselves.

Symbolic selfplay The Partner Planner achieves strong performance in symbolic selfplay, as shown in Table 6.2. The ablated version, which does not model unshared dots, also performs well, but worse than the full Partner Planner. This demonstrates the utility of modeling unshared perspective.

Both the full and ablated Partner Planner outperform the baseline of Fried et al. (2021) and coached human performance from the training data of Udagawa & Aizawa (2019), demonstrating the utility of partner modeling. Much of this success comes from the ability to control the belief entropy selection heuristic, as shown by the performance of the ablated Partner Planner over human performance. The selection heuristic encourages the Partner Planner to be more patient and gather more information before selecting than most human participants, reflected in the average number of turns per game.

6.7 Conclusion

This chapter presented a state tracking representation for a visually-grounded reference game, which explicitly captured perspective-dependent ambiguity. Experiments showed that combining this representation with a probabilistic model of partner responses allowed agents to continuously ask informative questions, resulting in higher success rates than supervised baselines.

In the next chapter, we will build upon the state tracking representation, adding in language understanding and generation to produce an agent capable of succeeding with human partners.

7

Symbolic Planning and Code Generation for Grounded Dialogue

7.1 Introduction

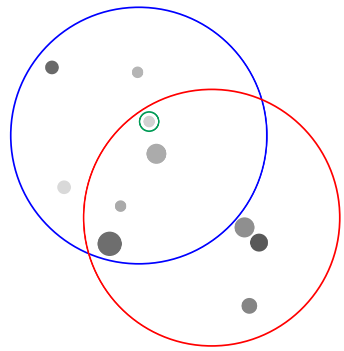
Our final chapter on state tracking representations proposes a full dialogue agent that successfully interacts with human partners. The previous chapter focused on symbolic planning. A full dialogue agent must not only be capable of planning but also interacting with human partners in natural language. This chapter closes that gap by extending the previous chapter's probabilistic state representation, which tracked uncertainty over task-specific progress measures, to a state representation that explicitly composes the semantics of utterances exchanged in the dialogue.

Success in grounded task-oriented dialogue requires intentional communication guided by strategic planning (Cohen & Perrault, 1979, Traum, 1994, Walker et al., 2004, Rieser & Lemon, 2009, FAIR et al., 2022, *inter alia*). Dialogue agents must read partner utterances, update their beliefs, then make a plan that furthers their goal. These plans must take into account both dialogue history and grounding, such as in an image. In end-to-end systems based solely on large language models (LLMs), this process is implicit and therefore difficult to control, requiring extra supervision (Christiano et al., 2023) or expensive search (Lu et al., 2022) to improve. While recent work has taken steps to rectify implicit reasoning via planning in language space, where intermediate steps are generated by an LLM (Wei et al., 2022), there is no guarantee that these approaches result in plans that further task progress. Additionally, planning in language space is expensive, requiring inference in an LLM (Yarats & Lewis, 2017, Guez et al., 2012).

Rather than implicit or heuristic reasoning, we are interested in explicit reasoning and planning over symbolic actions. Symbolic actions are controllable by construction, allowing system designers to easily build in task-specific knowledge (He et al., 2018, FAIR et al., 2022). This controllability is crucial for obtaining task-specific success using general tools, even with LLMs.

We provide an example from OneCommon, a particularly challenging grounded dialogue game (Udagawa & Aizawa, 2019). The goal of OneCommon is to, through dialogue, identify one dot in common with your partner, who has an overlapping but different view of an underlying set of dots, illustrated in Figure 7.1. The challenge in OneCommon is grounding the contextual spatial relationships described in language to dots.

Recent work has utilized code-generation for grounded language understanding



SPC:	Do you see a pair of dots, where the top left dot is medium-sized and grey and the bottom right dot is medium-sized and dark
Human:	No
SPC:	Do you see a pair of dots, where the top dot is small-sized and light and the bottom dot is medium-sized and grey
Human:	Yes I see them. Is there a small grey dot above the small light dot?
SPC:	No. Is there a large size and dark color dot to the left and below those?
Human:	Yes and there is a small grey dot below them as well for me.
SPC:	Yes. Let's select the small size and light color one.
SPC:	<select>
Human:	<select>

Figure 7.1: An example grounded dialogue from OneCommon. Our dialogue agent, SPC, and a human partner have different but overlapping circular views of a shared board. The agent and partner must collaborate through dialogue in order to find and select a shared dot. OneCommon demands careful, grounded reasoning.

(Dídac et al., 2023). In particular, they translate natural language questions to code as an intermediate representation, then execute that code to obtain an answer. Code has a couple appealing properties as an intermediate representation: First, modern language models are trained on a mixture of code and natural language, affording them the capability of, with some accuracy, translating between the two (Chen et al., 2021). Second, code acts as a compositional knowledge representation. This allows code-generation systems to perform grounded compositional reasoning, provided a library of Python functions that perform grounding (Liang et al., 2022).

We present a system, Symbolic Planning and Code-generation (SPC), that *reads* by translating partner utterances into code and *plans* based on symbolic reasoning over what to say next. Code as a compositional knowledge representation closely mirrors the compositional nature of utterances, which are composed of grounded parts. SPC plans by optimizing expected information gain, which has been shown to be effective at building a key aspect of collaborative dialogue: common ground (Yu et al., 2019, White et al., 2021, Chiu et al., 2022b). Symbolic planning allows SPC to explicitly and efficiently optimize for task success while taking advantage of task-specific properties.

We evaluate our SPC system on the most challenging subset of the OneCommon task, comparing our system to the previous state-of-the-art supervised system for the task (Fried et al., 2021). In both evaluations with human partners and automated self-play evaluations, we find that our approach substantially outperforms the previous state-of-the-art in task accuracy, improving from 56% to 69% accuracy, and obtains comparable task accuracy to human-human pairs on average.

7.2 Overview: Reference Games

Collaborative reference games pair an agent and a partner in order to build common ground through natural language dialogue (Haber et al., 2019, Khani et al., 2018, He et al., 2017, Udagawa & Aizawa, 2019). Mirroring realistic scenarios, many reference games are also partially observable, where the agent and partner have different perspectives, and so they must resolve ambiguity.

OneCommon (Udagawa & Aizawa, 2019), as shown in Figure 7.1, is a reference game that exemplifies two challenges: grounding and planning. In OneCommon, the agent and partner see different but overlapping views of a set of dots, and the goal is to find and select one dot common to both players' views. Grounding in OneCommon is particularly difficult due to the dot-based visual context, which requires abstract spatial reasoning. Planning is complicated by the partial observability caused by differing perspectives, which require agents to use complex referring expressions in order to avoid ambiguity.[1] We focus on OneCommon due to its simplicity and difficulty.

Our approach to grounded reference games separates symbolic reasoning from language, allowing explicit steering. Our system, Symbolic Planning and Code-generation (SPC), breaks down a turn into three procedures: reading, planning, and writing. Reading and writing convert from language to symbols and vice versa, while planning reasons in purely symbolic space.

The agent maintains a belief distribution over possible worlds, z, representing task-specific unknowns. The goal of dialogue is to gain information about z until the agent is confident enough to end the game. At each turn, the agent **reads** the partner's utterance u, converting it into a symbolic action, $p(x|u)$. This symbolic action potentially builds upon the action x' of a previous utterance, u'. The agent then **plans** in symbolic space. The system uses reasoning to update its belief state, $p(z|u) = \sum_x p(z|x)p(x|u)$, then produces a response y^* of what to say next, which it describes

[1]The contexts in OneCommon were constructed to make referring expressions challenging and context-dependent. For example, if the agent sees only light dots, a relatively 'dark' dot for the agent may not be considered dark at all by the partner.OneCommon is an ideal testbed for pragmatic methods that reason about contextual meaning. While our approach does not address pragmatics, we hope future work will.

in language to the partner.

In OneCommon, given a set of dots $\mathcal{D}$, the state $z \in \{0, 1\}^{|\mathcal{D}|}$ represents which dots the agent believes are contained (1) and not contained (0) in the partner's view, illustrated in Figure 7.3. We call a set of dots a *configuration*. The action representation of partner, x and x', and agent utterances, y^*, alike is also a configuration in $\{0, 1\}^{|\mathcal{D}|}$, as well as any answers or confirmations to previous questions.

7.3 Reading: From Language to Symbols

Reading in SPC requires interpreting utterances to a grounded symbolic action, which in turn facilitates the planning stage. Consider the following exchange:

> Agent: *Do you see a triangle of dark dots?*
> Partner: *Yes, is there a small grey one below it?*

Reading has several challenges. First, reading requires grounding utterances in context, e.g. the shapes and relations. Second, utterances are compositional. For example, the partner utterance builds on top of the previous utterance through coreference. Finally, a reading system must act quickly, as real-time dialogue systems require reasonable response times.

7.3.1 Code Generation

In SPC, reading is implemented as code generation. Given a dialogue, we generate Python code[2] which is then used as a meaning function to produce a distribution over all valid interpretations of the utterance's symbolic action (Figure 7.2). The code calls

[2]We target Python as our code representation since it is well-understood by large language models. However, in principle, our system could target other languages such as Prolog or SQL.

perceptual library functions with grounded semantics, drawn from a task-specific API. This perceptual library allows the system to both ground elements of the utterance and compositionally build upon previous utterances. Consider the following abbreviated example, based on OneCommon:

```
from perceptual_library import is_small, ...
dot1, dot2, dot3, ... = get_dots()
```

Agent: *Do you see a triangle of dark dots?*

```
agent_configs = set([
  Config(dot1, dot2, dot3),
  Config(dot3, dot4, dot1)
])
```

Partner: *Yes, is there a small grey one below it?*

```
def turn(prev_configs):
  configs = set()
  for prev_config in prev_configs:
    for dot in single_dots(exclude=prev_config):
      if (
        is_small(dot)
        and is_grey(dot)
        and is_below(dot, prev_config)
      ):
        configs.add(Config(dot, prev_config))
  return configs
partner_configs_x = turn(agent_configs)
```

The code in the meaning function is imperative, but represents a set of declarative constraints representing $p(x|u)$.[3] The meaning function for the partner turn, `turn(prev_configs)`, takes as input the distribution over symbolic actions of a previous turn, $p(x')$, and

[3]In OneCommon, the distribution over symbolic actions $p(x|u)$ is represented as represented as a categorical distribution over configurations with probabilities based on the size of the circumcircle.

yields a set of possible interpretations of the current turn, $p(x|u) = \sum_{x'} p(x|u, x')p(x')$.[4] Because utterances can have multiple valid interpretations due to ambiguity, `prev_configs` represents a distribution.[5]

Within `turn`, we consider all valid configurations while marginalizing over x', i.e. interpretations in `prev_configs`. For each interpretation, each `dot` is considered. If the new `dot` satisfies the semantics of the utterance, checked step-by-step via grounded perceptual library functions such as `is_small(d)`, then it is a valid interpretation of the current utterance and is used to create a new `Config`.

The perceptual library functions are drawn from a manually-defined library. For OneCommon, we define these functions using domain-specific knowledge:

```
def is_small(d): return d.size < -0.3
```

The perceptual library for OneCommon can be found here.

7.3.2 Prompting

Reading is implemented with large language model (LLM) code generation. While LLMs can generate accurate code, full code specifications (subsection 7.3.1) are lengthy and therefore too slow to generate for real-time use. We break down code generation into four steps, where some steps do not require any calls to an LLM. Decreasing the number of output tokens guarantees a speedup, assuming consistent latency. See the code for details on the code LLM and prompts we use.[6]

[4]The symbolic action of a previous turn x' may also depend on other previous utterances u'. For simplicity, we omit that in the notation.

[5]SPC is able to intentionally produce ambiguous descriptions if that improves task success, as illustrated in this example.

[6]We release the code here.

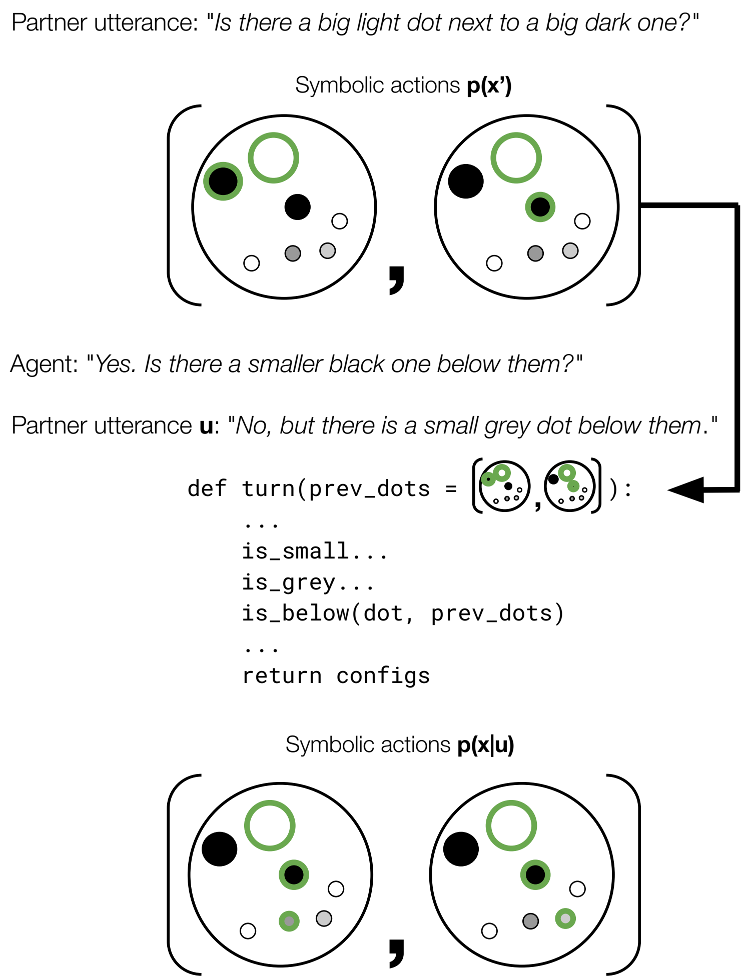


Figure 7.2: Overview of Reading. The generated meaning function for utterance u takes the previous symbolic action distribution $p(x')$ from a prior turn and yields the interpretations $p(x|u)$, using code as a compositional representation (section 7.3).

Dialogue Act: Classify partner utterances as one of three dialogue acts: Start a NEW line of questioning, ask a FOLLOW-UP question, END the dialogue.
Reference: Predict which previous turn x' the utterance is following up on, if any:

> Agent: *Do you see a triangle?*
> Partner: *Yes, is there a small grey dot below it?*
>
> ```
> dialogue act: follow-up
> refer: turn 1
> ```

The system grounds the dots mentioned in the previous turn: `agent_configs`, which is stored by the system. This allows referring to other turns besides the previous.
Constraint Generation: Predict the new dots mentioned in the partner utterance alongside code fragments that express the semantics, without the boilerplate code, in the example above:

> Partner: *Yes, is there a small grey one below it?*
>
> ```
> 1 new dot
> is_small(dot)
> is_grey(dot)
> is_below(dot, prev_dots)
> ```

Compose: Finally, we utilize a template to compose all of this information back into the full code representation for execution.

7.4 Planning: From Symbols to Responses

To perform well in collaborative reference games, it is essential to build common ground quickly and accurately by carefully reasoning about what information has been gathered so far, as well as what to say next. SPC addresses these desiderata by

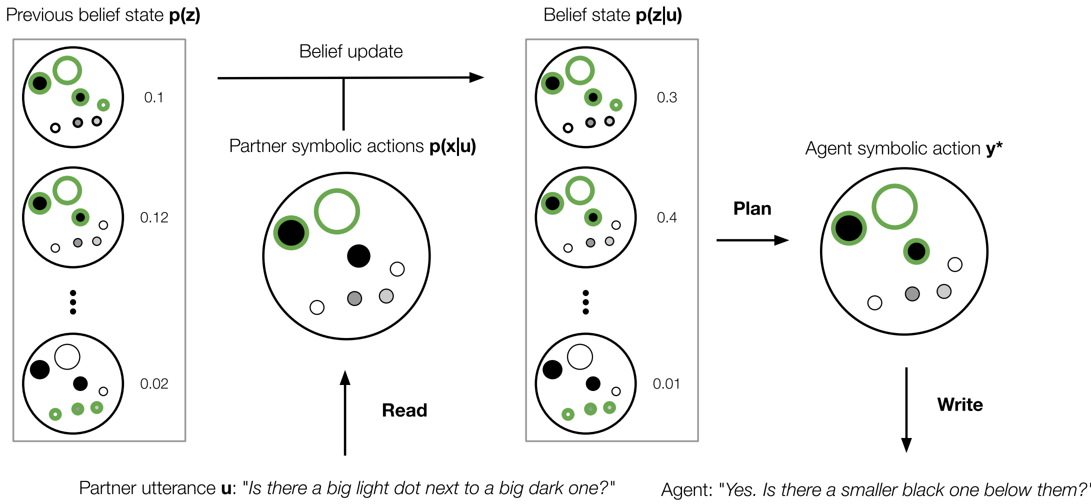


Figure 7.3: Overview of Planning. Partner utterances are interpreted by a meaning function generated by a code LLM (read), producing a distribution over valid symbolic interpretations, $p(x|u)$. This is used to symbolically update the belief state, $p(z|u)$, increasing the probability of worlds (shared dots) that are consistent with x. This belief state is used to symbolically plan the agent's next utterance, y^*, by optimizing the expected information gain, which is described to the partner (write).

planning in symbolic space, over the symbolic actions produced by reading.

We have two challenges: First, to incorporate the new information from the partner's utterance while accounting for task-specific grounding as well as dialogue history. Second, given this new information, the system must decide either to end the game or how to improve the probability of success.

Planning requires us to model the actions of the partner given the shared state. To do this we need task specific models of our partner, $p(x \mid z)$, and our partner's reponse to us, $p(x|z, y)$. In OneCommon, we use the models defined in Chapter 6.

7.4.1 Belief update

Starting from a prior over the previous belief $p(z)$, we incorporate probabilistic evidence from the utterance $p(x|u)$. This requires marginalizing over all valid symbolic actions x from the reading step. In practice, $p(x|u)$ is sparse, and symbols x with non-zero support are very similar. We therefore approximate this marginalization with a point estimate:

$$\begin{aligned} p(z|u) &= \sum_x p(z|x)p(x|u) \\ &= \sum_x \frac{p(x|z)p(z)}{p(x)} p(x|u) \\ &\approx \sum_x \frac{p(x|z)p(z)}{p(x)} 1(x = x^*) \\ &\propto p(x^* \mid z)p(z), \end{aligned} \tag{7.1}$$

where $x^* = \operatorname{argmax}_x p(x|u)$.

We give an example of this process in Figure 7.3. In this case, a 'big light dot next to a big dark one' could have two valid interpretations, the big light dot and the black dot to the left, or the other black dot to the right. We approximate this distribution with the most likely interpretation x^*. In OneCommon, we use the most compact[7] as x^*, yielding the black dot on the left. The belief state is then updated to $p(z|u)$, shown in Figure 7.3 (center).

7.4.2 Planning

Given the updated belief, SPC then plans its next action. The challenge here is to ensure task success, e.g. finding one dot in common. This requires both exploring by

[7]We define the compactness of a configuration as the radius of the circumcircle. An ideal approximation would take into account more context, such as the relative sizes.

building common ground, then exploiting that knowledge to win the game.

We formalize exploration as the expected information gain, a quantity that codifies how much the agent can expect to learn about possible worlds z after taking an action (Lindley, 1956). That action then elicits a response from the partner, providing information about the uncertain world state. For example, if the agent has asked about a set of dots and already received a 'no', then asking further questions about those dots would not reduce uncertainty.

Formally, we optimize

$$y^* = \operatorname*{argmax}_y H[z|u] - \mathbb{E}_{x_y|y}\left[H[z \mid u, y, x_y]\right], \tag{7.2}$$

where $H[z|u]$ is the entropy of the current belief[8] and $H[z \mid u, y, x_y]$ the entropy of the posterior distribution. This second term is the key part of the objective. Assuming that we take action y, the expectation considers all hypothetical future partner responses x_y. We are penalized if after seeing these responses, we are still uncertain about the common ground z. This objective therefore encourages actions that reduce uncertainty. [9]

SPC chooses to exploit and end the game with the following heuristic: If the system is confident in success, i.e. the probability of task success is greater than hyperparameter θ (set to 0.8), SPC ends the game.

[8]The belief entropy $H[z|u]$ in the definition of information gain is constant with respect to the plan x, and can be dropped from the objective.

[9]The distribution $p(x_y|y) = \sum_z p(x_y|y,z)p(z)$ also uses the partner response model $p(x_y|y,z)$.

7.4.3 Writing

We apply one of three templates to realize symbolic plans into utterances. The template is determined by the dialogue act and features of the dots in the plan:

1. start: `Do you see a pair of dots, where the {position} dot is {size}-sized and {color} and the {position} dot is {size}-sized and {color}?`
2. follow-up: `Is there a {size} size and {color} color dot {position} those?`
3. select: `Let's select the {size} size and {color} color one. <selection>`

7.5 Experimental Setup

We conduct two evaluations of SPC on the OneCommon task. We compare to the state-of-the-art baseline system of Fried et al. (2021), which we refer to as Imitate. Imitate is a pipelined system, where each part is fully supervised. Imitate uses a neural representation of dialogue history in combination with a neural-CRF reference resolution module to understand grounded language. In order to generate, Imitate relies on a pragmatic planning procedure, which plans in a mixture of symbolic and language space, prioritizing descriptions of dots that are easily understood.

We first perform human evaluation, evaluating the task success of systems when paired with human partners. This setting is challenging, requiring the system to handle both the linguistically diverse utterances and a range of strategies of human partners. We recruit 19 workers from Amazon's Mechanical Turk to play with one of three partners: SPC, the most successful past system for the task (Fried et al., 2021), or another human. We pay $15 per hour, with $1.00 per game at an average of 4 minutes per game. We additionally give a bonus of $0.15 for every game. We use 100 visual contexts from the most difficult[10] partition of OneCommon. We pay workers $\$1.00$ per game, with a $\$0.15$ bonus if they win. We collect 287 completed dialogues in total, where both players selected a dot.

We secondarily evaluate systems in self-play, where systems are paired with a copy of themselves. This isolates strategic efficiency by ensuring the agent's partner has the same skill as the agent. The 200 games share the same contexts across systems.

We include an additional system in self-play, GPT4 2-shot[11], which gets two full human dialogues as examples. Each human dialogue example starts with a description of the context the agent sees. The full prompts can be viewed here.

Parameterization For code generation in the reading phase, we use GPT-4[12] (OpenAI, 2023). The symbolic actions in OneCommon consist of sets of dots and confirmations, while the belief over symbolic states, $p(z)$, captures which dot configurations are shared and is designed to account for dot proximity. We use the same prior as Chapter 6, which parameterized an Ising model based on the pairwise distances between shared dots. The symbolic partner models, $p(x \mid z)$ and $p(x \mid y, z)$, are drawn from Chiu et al. (2022b), and incorporate a similar bias based on dot proximity.

7.6 Results

Human evaluation In human evaluation, SPC obtains substantially higher task accuracy than the baseline model of Fried et al. (2021), and is comparable to human performance on average. This demonstrates that the combination of symbolic

[10]The number of shared dots is four.
[11]We do not include GPT4 2-shot in human evaluation, as its self-play performance is poor.
[12]Specifically `gpt-4-0613`.

Agent	Success	Turns	Games
SPC	68.8%	7.77	96
Imitate	55.6%	6.61	117
Human	67.6%	5.03	74
Human†	65.8%	4.97	2,189

Table 7.1: The average success rate, average number of turns, and total number of games between agents and human partners on the hardest setting of OneCommon, with 4 shared dots. † indicates statistics from the OneCommon dataset (Udagawa & Aizawa, 2019).

information-gain planning and code-generation in SPC is more effective than the baseline's language-space planning objective and supervised reference resolution.

We see a more nuanced story when conducting a skill-based analysis of the human evaluation results, presented in Figure 7.4. A worker's skill is given by their average success rate with other human partners. The x-axis of the graph, the minimum success rate, increasingly filters workers from left to right: the left side of the graph shows all workers, while the far right shows only those workers who won nearly all of their human-human games. Skilled human partners have a higher success rate with other humans, as opposed to when partnered with SPC. Additionally, the success rate of SPC improves with human skill, while the success rate of human partners with the baseline system, Imitate, remains relatively constant across skill levels, implying that SPC is more responsive than the baseline to strategies used by humans.

SPC also takes more turns on average than both the baseline and human-human games. We hypothesize that this difference is caused by shorter human partner responses to the system, and therefore less information shared by the human partner. In Table 7.2, we confirm that the average and median number of words per human utterance are significantly lower for humans partnered with SPC than any other agent

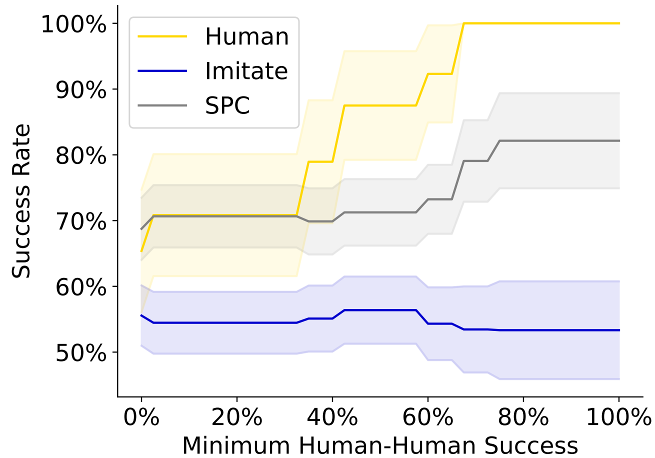


Figure 7.4: Success rate of the different agent types with human partners, with progressive filtering of human partners by their success rate along the x-axis. Shaded regions give standard errors.

Agent	Avg $\|u\|$	Median $\|u\|$
SPC	6.95	4
Imitate	9.62	8
Human	15.06	14

Table 7.2: The average and median number of words per utterance by human partners for different agent types in human evaluation.

type.

Self-play Similarly to human evaluation, SPC outperforms the baseline Imitate system in self-play as shown in Table 7.3. Compared to the baseline, SPC takes more turns on average, but has a higher success rate. We attribute both the longer games and higher success to symbolic planning, which ensures conservative playing. Interestingly, SPC self-play takes fewer turns on average than SPC-human pairings. We hypothesize that this is due to both copies of SPC communicating a consistent amount

Agent	Success	Avg # turns
SPC	84.0%	4.83
Imitate	63.5%	3.31
GPT4 2-shot	19.0%	9.26
Human†	65.8%	4.97

Table 7.3: The success rate of different agents in 200 self-play games on the hardest setting of OneCommon, with 4 shared dots. A higher success rate is better. The human performance is from the OneCommon dataset (Udagawa & Aizawa, 2019).

of information every turn. This also highlights the importance of human evaluation, which evaluates with a large population of partners.

We also find that GPT4 2-shot performs poorly in self-play. We attribute this to overly-agreeable responses, where the agents choose a dot without thorough verification or reasoning. This occurs despite the much longer dialogues, in comparison to all other agent types.

7.7 Analysis

Qualitative analysis We present a qualitative example of a dialogue between SPC and a human partner in Figure 7.5. This long dialogue illustrates the benefits of explicit belief updates and planning: The conversation starts off with many rejections, which the agent uses to deduce the shared configurations. Eventually, a common set of dots is found and the game ends in a success. Without explicit planning, it would have been unlikely for SPC to have succeeded at the end of the conversation.

Reading speed analysis We perform a speed ablation of the code-generation prompt in SPC. SPC uses a sequence of steps for reading, involving dialogue act classification, code fragment generation, and composing the full code representation based on the

Prompt style	Acc	Time (s)	Len
SPC	86.7%	5	36
Full	84.0%	18	176

Table 7.4: The average accuracy, speed, and output length (number of tokens) for the sequential and full code generation methods in our benchmark reading task.

output of these steps. We compare this to a prompt that generates the full meaning function.

We evaluate both of these prompts in a reading task, where the goal is to read utterances generated by SPC and recover the underlying plans, measured by accuracy. In Table 7.4, we see that both styles of prompts have similar similar accuracy, but the sequential, decomposed approach is much faster due to shorter outputs.

7.8 Related Work

Prior work on collaborative reference games focuses on building common ground (He et al., 2017, Haber et al., 2019, Khani et al., 2018). Prior work by Fried et al. (2021) implements an approximation of pragmatic reasoning on OneCommon, but plans in language space and utilizes supervised models for mapping language to symbols. Khani et al. (2018) plan in symbolic space, but without natural language. We plan in symbolic space and map from language to symbols via code generation.

Dialogue systems have a long history of reasoning with symbolic actions. When available, symbolic actions have been found to improve the performance of dialogue systems, especially in the setting of grounded dialogue (Winograd, 1971, Young, 2006, He et al., 2018, Andreas et al., 2020, FAIR et al., 2022). The closest work to ours is Cicero, which utilizes symbolic planning in a system for Diplomacy, a dialogue and strategy game that requires negotiation and coordination between players (FAIR et al., 2022). Cicero requires a supervised dataset to train their system. We use code LLMs which require minimal supervision beyond constructing a small perceptual grounding API.

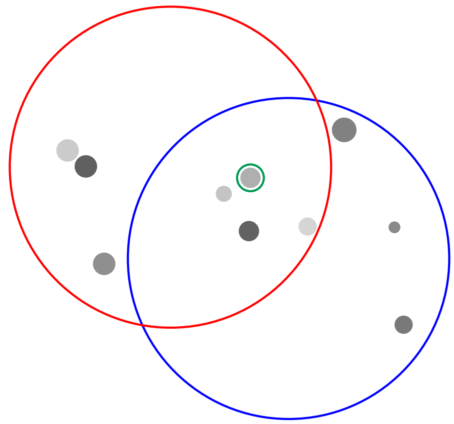
Planning in dialogue systems has recently eschewed symbolic actions in favor of planning directly in text, where systems either perform roll-outs, tree-search, or other forms of intermediate reasoning in language. This allows system designers to avoid manually defining symbolic actions (Yarats & Lewis, 2017, Jang et al., 2020, Gandhi et al., 2023). However, the accuracy of language-space planners is still low in many settings (Fried et al., 2021, Valmeekam et al., 2023). We focus on symbolic planning, where planning is defined in a space that ensures accuracy and controllability.

With the recent progress in large language modeling, code generation for modular grounded systems has quickly gained interest. Grounded code generation systems do not require task-specific training data, making them cheap to apply. A body of work utilizes a large language model for instruction following by generating Python code that makes calls to lower-level perception libraries (Liang et al., 2022, Dídac et al., 2023, Gupta & Kembhavi, 2022, Gao et al., 2023). This extends prior work on executable semantic parsing (Liang, 2016, Johnson et al., 2017, Cheng et al., 2018) with large language models. Concurrent work has also utilized code-generation to interpret language, integrated with symbolic reasoning (Wong et al., 2023). We apply these advances to the setting of grounded task-oriented dialogue, where code generation grounds language to symbolic actions for use in explicit planning.

7.9 Conclusion

This chapter presented a method that combines symbolic, probabilistic state representations with semantic parsing to obtain a strategic and interactive task-oriented dialogue system. The method, Symbolic Planning and Code-generation (SPC), separated symbolic reasoning from language by using an LLM to generate executable code functions which represent the meaning of utterances, mapping from language to symbolic actions for use in Bayesian planning. Despite using minimal supervision beyond a task-specific API and few-shot examples, SPC substantially outperformed a state-of-the-art system for the OneCommon task in both human evaluations and automatic self-play evaluations.

An exciting future direction is to automate the design of both code-based and probabilistic state representations. Our work represents a first step toward using general-purpose code as a representation for dialogue and interaction tasks. An ideal system will synthesize code-based representations that afford more flexible interaction with people, e.g., represent a broader range of user actions, both linguistic and grounded, to construct broadly useful interactive systems. Additionally, systems should propose and formalize the probabilistic belief state, as well as create and train partner models to plan accurately.



SPC:	Do you see a pair of dots, where the bottom dot is medium-sized and dark and the top dot is small-sized and light
Human:	No. Do you see a pair of medium sized dots, close together, one is dark grey the other light grey. The light grey one is slightly above and the left of the dark one.
SPC:	No. Do you see a pair of dots, where the left dot is medium-sized and dark and the right dot is medium-sized and light
Human:	No, do you see a lone medium sized grey dot?
SPC:	No. Do you see a pair of dots, where the left dot is medium-sized and light and the right dot is small-sized and grey
Human:	No. do you see a pair where the right one is medium and grey and the left one is smaller and lighter. The smaller one is slightly below the medium sized one.
SPC:	Yes. Is there a medium size and light color dot to the right and below those?
Human:	Yes
SPC:	Let’s select the medium size and grey color one.
Human:	<select>
SPC:	<select>

Figure 7.5: An example dialogue between SPC and a human partner.

8

Conclusion

This thesis explored state tracking representations for three applications: language modeling, question answering, and dialogue. Through the lens of latent variable models, we scaled tractable and uncertainty-aware state tracking representations for each application. For language modeling, we imposed constraints that allowed the discrete state tracking representations to scale to large sizes (Chapters 3 and 4). For question answering, we employed a hierarchical choice model composed of LLMs that used automatic domain expertise to enable accurate state tracking (Chapter 5). For dialogue, we utilized large language models to produce executable state representations that allowed agents to effectively plan when composed with a probabilistic and symbolic state representation (Chapters 6 and 7). In each case, we found that tractable state tracking representations led to surprisingly accurate decisions.

The methods presented in this thesis relied on latent variable models to inject domain expertise at the cost of computational complexity. Domain expertise enabled us to craft tractable models that were interpretable to humans (Chapters 3 and 5) and controllable (Chapters 6 and 7). However, this is potentially misaligned with the Bitter Lesson (Sutton, 2019). A recent trend in NLP is to rely on large corpora to teach models domain expertise (OpenAI, 2023). This leads to a natural question: What will the methods explored in this thesis look like when domain expertise is captured by LLMs?

We offer three perspectives on this question. First, many decision-making processes involve reasoning over latent variables, which may not be observable. This can lead to the impossibility of learning domain expertise from only observed data (Kumor et al., 2021). In such settings, system-based pipeline approaches such as those presented in Liu et al. (2024) and Chapter 7 offer a solution when naive approaches with domain-agnostic LLMs fail. Second, creating new knowledge from current domain expertise is naturally a latent variable problem. In Chapter 5, we showed that a latent variable approach can be used to recover the steps that lead from a question to an answer. An extension of this work could be used to learn how to generate answers to complex questions with unknown answers. Finally, we hope that AI agents with domain expertise can inject their domain knoweldge into the systems explored in this thesis without human intervention. Recent progress in software engineering agents (Yang et al., 2024, Shi et al., 2024) hints at the potential of such systems. Future work should strive towards autonomous agents that search for robust neurosymbolic systems that design and test formal frameworks themselves.

We conclude this thesis with a call to action. We believe that a combination of the three main ideas of this thesis will push autonomous agents forward. Trustworthy autonomous agents will plan with symbolic objectives, produced by LLMs but verified by humans, and solved by model-based planning that reasons over uncertain world state or alternative worlds (chapter 6). The planning world model will consider alternative worlds, produced by a LLM that can apply domain knowledge to narrow the number of alternative worlds to allow for efficient sparse inference (chapter 5). LLMs will be trained directly for recalling alternative worlds via the methods explored in chapters 3 and 4. The framework of latent variable models provides a natural abstraction with which to reason about each of these components – but also how to optimize them jointly. We anticipate autonomous agents composed of modular systems for state tracking and decision-making, trained jointly in an end-to-end and data-driven system.

References

Andreas, J., Bufe, J., Burkett, D., Chen, C., Clausman, J., Crawford, J., Crim, K., DeLoach, J., Dorner, L., Eisner, J., Fang, H., Guo, A., Hall, D., Hayes, K., Hill, K., Ho, D., Iwaszuk, W., Jha, S., Klein, D., Krishnamurthy, J., Lanman, T., Liang, P., Lin, C. H., Lintsbakh, I., McGovern, A., Nisnevich, A., Pauls, A., Petters, D., Read, B., Roth, D., Roy, S., Rusak, J., Short, B., Slomin, D., Snyder, B., Striplin, S., Su, Y., Tellman, Z., Thomson, S., Vorobev, A., Witoszko, I., Wolfe, J. A., Wray, A., Zhang, Y., & Zotov, A. (2020). Task-oriented dialogue as dataflow synthesis. *CoRR*, abs/2009.11423.

Atanasova, P., Simonsen, J. G., Lioma, C., & Augenstein, I. (2022). Diagnostics-guided explanation generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36 (pp. 10445–10453).

Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In Y. Bengio & Y. LeCun (Eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Bard, N., Foerster, J. N., Chandar, S., Burch, N., Lanctot, M., Song, H. F., Parisotto, E., Dumoulin, V., Moitra, S., Hughes, E., Dunning, I., Mourad, S., Larochelle, H., Bellemare, M. G., & Bowling, M. (2020). The hanabi challenge: A new frontier for ai research. *Artificial Intelligence*, 280.

Bastings, J., Aziz, W., & Titov, I. (2019). Interpretable neural predictions with differentiable binary variables. In A. Korhonen, D. Traum, & L. Màrquez (Eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* (pp. 2963–2977). Florence, Italy: Association for Computational Linguistics.

Bengio, Y., Ducharme, R., Vincent, P., & Janvin, C. (2003). A neural probabilistic language model. *J. Mach. Learn. Res.*, 3, 1137–1155.

Bottou, L. & LeCun, Y. (2005). Graph transformer networks for image recognition.

Boulanger-Lewandowski, N., Bengio, Y., & Vincent, P. (2012). Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12 (pp. 1881–1888). Madison, WI, USA: Omnipress.

Bradbury, J., Merity, S., Xiong, C., & Socher, R. (2016). Quasi-recurrent neural networks. *CoRR*, abs/1611.01576.

Brown, P. F., deSouza, P. V., Mercer, R. L., Pietra, V. J. D., & Lai, J. C. (1992). Class-based n-gram models of natural language. *Comput. Linguist.*, 18(4), 467–479.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33, 1877–1901.

Brown-Schmidt, S. & Hanna, J. E. (2011). Talking in another person's shoes: Incremental perspective-taking in language processing. *Dialogue Discourse*, 2, 11–33.

Buys, J., Bisk, Y., & Choi, Y. (2018). Bridging hmms and rnns through architectural transformations.

Chen, H. & Ji, Y. (2020). Learning variational word masks to improve the interpretability of neural text classifiers. In B. Webber, T. Cohn, Y. He, & Y. Liu (Eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 4236–4251). Online: Association for Computational Linguistics.

Chen, J., Lin, S.-t., & Durrett, G. (2019). Multi-hop question answering via reasoning chains. *arXiv preprint arXiv:1910.02610*.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., & Zaremba, W. (2021). Evaluating large language models trained on code.

Cheng, J., Reddy, S., Saraswat, V., & Lapata, M. (2018). Learning an executable neural semantic parser.

Chiang, D. & Riley, D. (2020). Factor graph grammars. *arXiv preprint arXiv:2010.12048*.

Chiu, J., Deng, Y., & Rush, A. (2021). Low-rank constraints for fast inference in structured models. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, & J. W. Vaughan (Eds.), *Advances in Neural Information Processing Systems*, volume 34 (pp. 2887–2898).: Curran Associates, Inc.

Chiu, J. & Rush, A. (2020a). Scaling hidden Markov language models. In B. Webber, T. Cohn, Y. He, & Y. Liu (Eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1341–1349). Online: Association for Computational Linguistics.

Chiu, J. & Rush, A. (2020b). Scaling hidden Markov language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1341–1349). Online: Association for Computational Linguistics.

Chiu, J., Zhao, W., Chen, D., Vaduguru, S., Rush, A., & Fried, D. (2023). Symbolic planning and code generation for grounded dialogue. In H. Bouamor, J. Pino, & K. Bali (Eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing* (pp. 7426–7436). Singapore: Association for Computational Linguistics.

Chiu, J. T., Zhao, W., Fried, D., & Rush, A. M. (2022a). Modeling perspective-dependent ambiguity in collaborative dialogue. In *The Third Wordplay: When Language Meets Games Workshop*.

Chiu, J. T., Zhao, W., Fried, D., & Rush, A. M. (2022b). Modeling perspective-dependent ambiguity in collaborative dialogue. In *The Third Wordplay: When Language Meets Games Workshop*.

Choi, Y., Vergari, A., & den Broeck, G. V. (2020). Probabilistic circuits: A unifying framework for tractable probabilistic models.

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., Belanger, D., Colwell, L., & Weller, A. (2020). Rethinking attention with performers.

Christiano, P., Leike, J., Brown, T. B., Martic, M., Legg, S., & Amodei, D. (2023). Deep reinforcement learning from human preferences.

Clark, H. H. & Marshall, C. R. (1981). Definite knowledge and mutual knowledge. In A. K. Joshi, B. L. Webber, & I. A. Sag (Eds.), *Elements of Discourse Understanding* (pp. 10–63). Cambridge, UK: Cambridge University Press.

Clark, H. H., Schreuder, R., & Buttrick, S. (1983). Common ground at the understanding of demonstrative reference. *Journal of Verbal Learning and Verbal Behavior*, 22(2), 245–258.

Cohen, P. R. & Perrault, C. R. (1979). Elements of a plan-based theory of speech acts. *Cognitive Science*.

Corona Rodriguez, R., Alaniz, S., & Akata, Z. (2019). Modeling conceptual understanding in image reference games. In H. Wallach, H. Larochelle, A. Beygelzimer,

F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems*, volume 32: Curran Associates, Inc.

Dale, R. & Reiter, E. (1995). Computational interpretations of the gricean maxims in the generation of referring expressions. *Cognitive Science*, 19(2).

Dale, R. & Viethen, J. (2009). Referring expression generation through attribute-based heuristics. In E. Krahmer & M. Theune (Eds.), *Proceedings of the 12th European Workshop on Natural Language Generation (ENLG 2009)* (pp. 58–65). Athens, Greece: Association for Computational Linguistics.

Dedieu, A., Gothoskar, N., Swingle, S., Lehrach, W., Lázaro-Gredilla, M., & George, D. (2019). Learning higher-order sequential structure with cloned hmms.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, & T. Solorio (Eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (pp. 4171–4186). Minneapolis, Minnesota: Association for Computational Linguistics.

DeYoung, J., Jain, S., Rajani, N. F., Lehman, E., Xiong, C., Socher, R., & Wallace, B. C. (2020). ERASER: A benchmark to evaluate rationalized NLP models. In D. Jurafsky, J. Chai, N. Schluter, & J. Tetreault (Eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (pp. 4443–4458). Online: Association for Computational Linguistics.

Dídac, S., Menon, S., & Vondrick, C. (2023). Vipergpt: Visual inference via python execution for reasoning. *arXiv preprint arXiv:2303.08128*.

Doğan, F. I., Gillet, S., Carter, E. J., & Leite, I. (2020). The impact of adding perspective-taking to spatial referencing during human–robot interaction. *Robotics and Autonomous Systems*, 134, 103654.

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

FAIR, M. F. A. R. D. T., Bakhtin, A., Brown, N., Dinan, E., Farina, G., Flaherty, C., Fried, D., Goff, A., Gray, J., Hu, H., Jacob, A. P., Komeili, M., Konath, K., Kwon, M., Lerer, A., Lewis, M., Miller, A. H., Mitts, S., Renduchintala, A., Roller, S., Rowe, D., Shi, W., Spisak, J., Wei, A., Wu, D., Zhang, H., & Zijlstra, M. (2022). Human-level play in the game of *diplomacy* by combining language models with strategic reasoning. *Science*, 378(6624), 1067–1074.

Fang, Y., Sun, S., Gan, Z., Pillai, R., Wang, S., & Liu, J. (2020). Hierarchical graph network for multi-hop question answering. In B. Webber, T. Cohn, Y. He, & Y. Liu (Eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 8823–8838). Online: Association for Computational Linguistics.

Felzenszwalb, P., Huttenlocher, D., & Kleinberg, J. (2004). Fast algorithms for large-state-space hmms with applications to web usage analysis. In S. Thrun, L. Saul, & B. Schölkopf (Eds.), *Advances in Neural Information Processing Systems*, volume 16: MIT Press.

Fraccaro, M., Sønderby, S. K., Paquet, U., & Winther, O. (2016). Sequential neural models with stochastic layers.

Fried, D., Alayrac, J.-B., Blunsom, P., Dyer, C., Clark, S., & Nematzadeh, A. (2020). Learning to segment actions from observation and narration. *arXiv preprint arXiv:2005.03684*.

Fried, D., Chiu, J. T., & Klein, D. (2021). Reference-centric models for grounded collaborative dialogue. In *Proceedings of EMNLP*.

Gan, Z., Li, C., Henao, R., Carlson, D., & Carin, L. (2015). Deep temporal sigmoid belief networks for sequence modeling.

Gandhi, K., Sadigh, D., & Goodman, N. D. (2023). Strategic reasoning with language models.

Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., & Neubig, G. (2023). Pal: Program-aided language models.

Gelman, A., Carlin, J. B., Stern, H. S., & Rubin, D. B. (2004). *Bayesian Data Analysis*. Chapman and Hall/CRC, 2nd ed. edition.

Geva, M., Khashabi, D., Segal, E., Khot, T., Roth, D., & Berant, J. (2021). Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9, 346–361.

Glockner, M., Habernal, I., & Gurevych, I. (2020). Why do you think that? exploring faithful sentence-level rationales without supervision. In T. Cohn, Y. He, & Y. Liu (Eds.), *Findings of the Association for Computational Linguistics: EMNLP 2020* (pp. 1080–1095). Online: Association for Computational Linguistics.

Glorot, X. & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249–256).: JMLR Workshop and Conference Proceedings.

Groeneveld, D., Khot, T., Mausam, & Sabharwal, A. (2020). A simple yet strong pipeline for HotpotQA. In B. Webber, T. Cohn, Y. He, & Y. Liu (Eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 8839–8845). Online: Association for Computational Linguistics.

Gu, S., Ghahramani, Z., & Turner, R. E. (2015). Neural adaptive sequential monte carlo.

Guez, A., Silver, D., & Dayan, P. (2012). Efficient bayes-adaptive reinforcement learning using sample-based search. *CoRR*, abs/1205.3109.

Gupta, T. & Kembhavi, A. (2022). Visual programming: Compositional visual reasoning without training.

Haber, J., Baumgärtner, T., Takmaz, E., Gelderloos, L., Bruni, E., & Fernández, R. (2019). The photobook dataset: Building common ground through visually-grounded dialogue. *CoRR*, abs/1906.01530.

Han, W., Jiang, Y., & Tu, K. (2017). Dependency grammar induction with neural lexicalization and big training data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing* (pp. 1683–1688). Copenhagen, Denmark: Association for Computational Linguistics.

He, H., Balakrishnan, A., Eric, M., & Liang, P. (2017). Learning symmetric collaborative dialogue agents with dynamic knowledge graph embeddings. *CoRR*, abs/1704.07130.

He, H., Chen, D., Balakrishnan, A., & Liang, P. (2018). Decoupling strategy and generation in negotiation dialogues. *CoRR*, abs/1808.09637.

Heafield, K., Pouzyrevsky, I., Clark, J. H., & Koehn, P. (2013). Scalable modified Kneser-Ney language model estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* (pp. 690–696). Sofia, Bulgaria: Association for Computational Linguistics.

Heeman, P. A. & Hirst, G. (1995). Collaborating on referring expressions. *Computational Linguistics*, 21(3), 351–382.

Heitmann, M., Siebert, C., Hartmann, J., & Schamp, C. (2020). More than a feeling: Benchmarks for sentiment analysis accuracy. *Available at SSRN 3489963*.

Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8), 1735–1780.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Rae, J. W., Vinyals, O., & Sifre, L. (2022). Training compute-optimal large language models.

Huang, L. & Chiang, D. (2005). Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology* (pp. 53–64).

Huang, Z. (2011). *Modeling Dependencies in Natural Languages with Latent Variables*. PhD thesis, University of Maryland.

Jang, Y., Lee, J., & Kim, K. (2020). Bayes-adaptive monte-carlo planning and learning for goal-oriented dialogues. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020* (pp. 7994–8001).: AAAI Press.

Javidian, M. A., Wang, Z., Lu, L., & Valtorta, M. (2020). On a hypergraph probabilistic graphical model. *Annals of Mathematics and Artificial Intelligence*, 88(9), 1003–1033.

Jin, S., Wiseman, S., Stratos, K., & Livescu, K. (2020). Discrete latent variable representations for low-resource text classification. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (pp. 4831–4842). Online: Association for Computational Linguistics.

Johnson, J., Hariharan, B., van der Maaten, L., Hoffman, J., Fei-Fei, L., Zitnick, C. L., & Girshick, R. (2017). Inferring and executing programs for visual reasoning.

Jordan, P. W. & Walker, M. A. (2005). Learning content selection rules for generating object descriptions in dialogue. *Journal of Artificial Intelligence Research*, 24(1).

Joshi, M., Chen, D., Liu, Y., Weld, D. S., Zettlemoyer, L., & Levy, O. (2020). SpanBERT: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8, 64–77.

Katharopoulos, A., Vyas, A., Pappas, N., & Fleuret, F. (2020). Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Keysar, B., Barr, D. J., Balin, J. A., & Brauner, J. S. (2000). Taking perspective in conversation: The role of mutual knowledge in comprehension. *Psychological Science*, 11(1), 32–38. PMID: 11228840.

Khani, F., Goodman, N., & Liang, P. (2018). Planning, inference and pragmatics in sequential language games. *Transactions of the Association for Computational Linguistics*, 6, 543–555.

Khashabi, D., Chaturvedi, S., Roth, M., Upadhyay, S., & Roth, D. (2018). Looking beyond the surface: A challenge set for reading comprehension over multiple sentences. In M. Walker, H. Ji, & A. Stent (Eds.), *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)* (pp. 252–262). New Orleans, Louisiana: Association for Computational Linguistics.

Kim, Y., Dyer, C., & Rush, A. M. (2019). Compound probabilistic context-free grammars for grammar induction. *CoRR*, abs/1906.10225.

Kingma, D. P. & Ba, J. (2017). Adam: A method for stochastic optimization.

Klein, D. & Manning, C. D. (2004). Parsing and hypergraphs. In *New developments in parsing technology* (pp. 351–372). Springer.

Krahmer, E. & van Deemter, K. (2012). Computational generation of referring expressions: A survey. *Computational Linguistics*, 38(1), 173–218.

Krakovna, V. & Doshi-Velez, F. (2016). Increasing the interpretability of recurrent neural networks using hidden markov models.

Kuhn, T., Niemann, H., & Schukat-Talamazzini, E. G. (1994). Ergodic hidden markov models and polygrams for language modeling. (pp. 357–360).

Kumor, D., Zhang, J., & Bareinboim, E. (2021). Sequential causal imitation learning with unobserved confounders. In A. Beygelzimer, Y. Dauphin, P. Liang, & J. W. Vaughan (Eds.), *Advances in Neural Information Processing Systems*.

Ladner, R. E. & Fischer, M. J. (1980). Parallel prefix computation. *J. ACM*, 27(4), 831–838.

Lai, G., Xie, Q., Liu, H., Yang, Y., & Hovy, E. (2017). RACE: Large-scale ReAding comprehension dataset from examinations. In M. Palmer, R. Hwa, & S. Riedel (Eds.), *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing* (pp. 785–794). Copenhagen, Denmark: Association for Computational Linguistics.

Lei, T., Barzilay, R., & Jaakkola, T. (2016). Rationalizing neural predictions. In J. Su, K. Duh, & X. Carreras (Eds.), *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (pp. 107–117). Austin, Texas: Association for Computational Linguistics.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., & Zettlemoyer, L. (2020a). BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In D. Jurafsky, J. Chai, N. Schluter, & J. Tetreault (Eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (pp. 7871–7880). Online: Association for Computational Linguistics.

Lewis, M., Yarats, D., Dauphin, Y. N., Parikh, D., & Batra, D. (2017). Deal or no deal? end-to-end learning for negotiation dialogues. *CoRR*, abs/1706.05125.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., & Kiela, D. (2020b). Retrieval-augmented generation for knowledge-intensive nlp tasks. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in Neural Information Processing Systems*, volume 33 (pp. 9459–9474).: Curran Associates, Inc.

Li, X.-Y., Lei, W.-J., & Yang, Y.-B. (2022). From easy to hard: Two-stage selector and reader for multi-hop question answering. *arXiv preprint arXiv:2205.11729*.

Liang, J., Huang, W., Xia, F., Xu, P., Hausman, K., Ichter, B., Florence, P., & Zeng, A. (2022). Code as policies: Language model programs for embodied control. In *arXiv preprint arXiv:2209.07753*.

Liang, P. (2005). Semi-supervised learning for natural language. In *MASTER'S THESIS, MIT*.

Liang, P. (2016). Learning executable semantic parsers for natural language understanding.

Lindley, D. V. (1956). On a Measure of the Information Provided by an Experiment. *The Annals of Mathematical Statistics*, 27(4), 986 – 1005.

Liu, M., Xiao, C., & Chen, C. (2021). Perspective-corrected spatial referring expression generation for human-robot interaction. *CoRR*, abs/2104.01558.

Liu, O., Fu, D., Yogatama, D., & Neiswanger, W. (2024). Dellma: A framework for decision making under uncertainty with large language models.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019a). Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019b). Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692.

Loshchilov, I. & Hutter, F. (2017). Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101.

Lu, X., Welleck, S., West, P., Jiang, L., Kasai, J., Khashabi, D., Bras, R. L., Qin, L., Yu, Y., Zellers, R., Smith, N. A., & Choi, Y. (2022). Neurologic a*esque decoding: Constrained text generation with lookahead heuristics.

Mao, J., Huang, J., Toshev, A., Camburu, O., Yuille, A. L., & Murphy, K. (2016). Generation and comprehension of unambiguous object descriptions. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*.

Marcus, M. P., Santorini, B., & Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2), 313–330.

Merialdo, B. (1994). Tagging English text with a probabilistic model. *Computational Linguistics*, 20(2), 155–171.

Merity, S., Keskar, N. S., & Socher, R. (2017). Regularizing and optimizing LSTM language models. *CoRR*, abs/1708.02182.

Merity, S., Xiong, C., Bradbury, J., & Socher, R. (2016). Pointer sentinel mixture models. *CoRR*, abs/1609.07843.

Mikolov, T., Deoras, A., Kombrink, S., Burget, L., & Cernocký, J. (2011). Empirical evaluation and combination of advanced language modeling techniques. (pp. 605–608).

Mikolov, T. & Zweig, G. (2012). Context dependent recurrent neural network language model. In *2012 IEEE Spoken Language Technology Workshop (SLT)* (pp. 234–239).

Narang, S., Raffel, C., Lee, K., Roberts, A., Fiedel, N., & Malkan, K. (2020). Wt5?! training text-to-text models to explain their predictions. *CoRR*, abs/2004.14546.

OpenAI (2023). Gpt-4 technical report.

Pearl, J. (1984). *Heuristics: intelligent search strategies for computer problem solving*. USA: Addison-Wesley Longman Publishing Co., Inc.

Peng, H., Pappas, N., Yogatama, D., Schwartz, R., Smith, N., & Kong, L. (2021). Random feature attention. In *International Conference on Learning Representations*.

Petrov, S., Barrett, L., Thibaux, R., & Klein, D. (2006). Learning accurate, compact, and interpretable tree annotation. (pp. 433–440).

Qi, P., Lin, X., Mehr, L., Wang, Z., & Manning, C. D. (2019). Answering complex open-domain questions through iterative query generation. In K. Inui, J. Jiang, V. Ng, & X. Wan (Eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* (pp. 2590–2602). Hong Kong, China: Association for Computational Linguistics.

Rabusseau, G., Balle, B., & Cohen, S. B. (2015). Weighted tree automata approximation by singular value truncation. *CoRR*, abs/1511.01442.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140), 1–67.

Rao, S. & III, H. D. (2018). Learning to ask good questions: Ranking clarification questions using neural expected value of perfect information. *CoRR*, abs/1805.04655.

Richardson, M., Burges, C. J., & Renshaw, E. (2013). MCTest: A challenge dataset for the open-domain machine comprehension of text. In D. Yarowsky, T. Baldwin, A. Korhonen, K. Livescu, & S. Bethard (Eds.), *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing* (pp. 193–203). Seattle, Washington, USA: Association for Computational Linguistics.

Rieser, V. & Lemon, O. (2009). Natural language generation as planning under uncertainty for spoken dialogue systems. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)* (pp. 683–691). Athens, Greece: Association for Computational Linguistics.

Ross, A., Chen, N., Hang, E. Z., Glassman, E. L., & Doshi-Velez, F. (2021). Evaluating the interpretability of generative models by interactive reconstruction. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21 New York, NY, USA: Association for Computing Machinery.

Roweis, S. (2000). Constrained hidden markov models. In S. Solla, T. Leen, & K. Müller (Eds.), *Advances in Neural Information Processing Systems*, volume 12: MIT Press.

Ruoss, A., Delétang, G., Medapati, S., Grau-Moya, J., Wenliang, L. K., Catt, E., Reid, J., & Genewein, T. (2024). Grandmaster-level chess without search.

Russell, S. & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition.

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108.

Shen, Y., Lin, Z., wei Huang, C., & Courville, A. (2018). Neural language modeling by jointly learning syntax and lexicon. In *International Conference on Learning Representations*.

Shen, Y., Tan, S., Sordoni, A., & Courville, A. (2019). Ordered neurons: Integrating tree structures into recurrent neural networks. In *International Conference on Learning Representations*.

Shi, Q., Tang, M., Narasimhan, K., & Yao, S. (2024). Can language models solve olympiad programming?

Shridhar, M. & Hsu, D. (2018). Interactive visual grounding of referring expressions for human-robot interaction. *CoRR*, abs/1806.03831.

Siddiqi, S. M., Boots, B., & Gordon, G. J. (2009). Reduced-rank hidden markov models. *CoRR*, abs/0910.0902.

Siddiqi, S. M. & Moore, A. W. (2005). Fast inference and learning in large-state-space hmms. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05 (pp. 800–807). New York, NY, USA: Association for Computing Machinery.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T. P., Simonyan, K., & Hassabis, D. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815.

Song, K., Jang, J., Shin, S., & Moon, I. (2019). Bivariate beta LSTM. *CoRR*, abs/1905.10521.

Stalnaker, R. C. (1970). Pragmatics. *Synthese*, 22(1-2), 272–289.

Stratonovich, R. L. (1959). Optimum nonlinear systems which bring about a separation of a signal with constant parameters from noise. In *Radiofizika*.

Sutton, R. (2019). The bitter lesson. *Incomplete Ideas (blog)*, 13(1), 38.

Takmaz, E., Giulianelli, M., Pezzelle, S., Sinclair, A., & Fernández, R. (2020). Refer, Reuse, Reduce: Generating Subsequent References in Visual and Conversational Contexts. In B. Webber, T. Cohn, Y. He, & Y. Liu (Eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 4350–4368). Online: Association for Computational Linguistics.

Thai, D., Ramesh, S. H., Murty, S., Vilnis, L., & McCallum, A. (2018). Embedded-state latent conditional random fields for sequence labeling. In *Proceedings of the 22nd Conference on Computational Natural Language Learning* (pp. 1–10). Brussels, Belgium: Association for Computational Linguistics.

Thayaparan, M., Valentino, M., & Freitas, A. (2020). A survey on explainability in machine reading comprehension. *arXiv preprint arXiv:2010.00389*.

Thomson, B. & Young, S. (2010). Bayesian update of dialogue state: A pomdp framework for spoken dialogue systems. *Comput. Speech Lang.*, 24(4), 562–588.

Thorne, J., Vlachos, A., Christodoulopoulos, C., & Mittal, A. (2018). FEVER: a large-scale dataset for fact extraction and VERification. In M. Walker, H. Ji, & A. Stent (Eds.), *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1* (*Long Papers*) (pp. 809–819). New Orleans, Louisiana: Association for Computational Linguistics.

Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic robotics*. Cambridge, Mass.: MIT Press.

Tran, K. M., Bisk, Y., Vaswani, A., Marcu, D., & Knight, K. (2016). Unsupervised neural hidden markov models. *CoRR*, abs/1609.09007.

Traum, D. R. (1994). *A Computational Theory of Grounding in Natural Language Conversation.* Technical report, Rochester University Department of Computer Science.

Trivedi, H., Balasubramanian, N., Khot, T., & Sabharwal, A. (2022). MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*.

Tu, M., Huang, K., Wang, G., Huang, J., He, X., & Zhou, B. (2020). Select, answer and explain: Interpretable multi-hop reading comprehension over multiple documents. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34 (pp. 9073–9080).

Tu, M., Wang, G., Huang, J., Tang, Y., He, X., & Zhou, B. (2019). Multi-hop reading comprehension across multiple documents by reasoning over heterogeneous graphs. In A. Korhonen, D. Traum, & L. Màrquez (Eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* (pp. 2704–2713). Florence, Italy: Association for Computational Linguistics.

Udagawa, T. & Aizawa, A. (2019). A natural language corpus of common grounding under continuous and partially-observable context. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01), 7120–7127.

Valmeekam, K., Sreedharan, S., Marquez, M., Olmo, A., & Kambhampati, S. (2023). On the planning abilities of large language models (a critical investigation with a proposed benchmark).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., & Polosukhin, I. (2017). Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems*, volume 30: Curran Associates, Inc.

Vieira, T. (2014). Gumbel-max trick and weighted reservoir sampling.

Vogel, S., Ney, H., & Tillmann, C. (1996). Hmm-based word alignment in statistical translation. In *Proceedings of the 16th Conference on Computational Linguistics - Volume 2*, COLING '96 (pp. 836–841). USA: Association for Computational Linguistics.

Walker, M. A., Whittaker, S. J., Stent, A., Maloor, P., Moore, J., Johnston, M., & Vasireddy, G. (2004). Generation and evaluation of user tailored responses in multimodal dialogue. *Cognitive Science*, 28(5), 811–840.

Wang, Z., Ma, Y., Liu, Z., & Tang, J. (2019). R-transformer: Recurrent neural network enhanced transformer.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E. H., Le, Q., & Zhou, D. (2022). Chain of thought prompting elicits reasoning in large language models. *CoRR*, abs/2201.11903.

Welbl, J., Stenetorp, P., & Riedel, S. (2018). Constructing datasets for multi-hop reading comprehension across documents. *Transactions of the Association for Computational Linguistics*, 6, 287–302.

White, J., Poesia, G., Hawkins, R., Sadigh, D., & Goodman, N. (2021). Open-domain clarification question generation without question examples. In M.-F. Moens, X. Huang, L. Specia, & S. W.-t. Yih (Eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing* (pp. 563–570). Online and Punta Cana, Dominican Republic: Association for Computational Linguistics.

Wiegreffe, S. & Marasovic, A. (2021). Teach me to explain: A review of datasets for explainable natural language processing. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

Williams, J., Raux, A., Ramachandran, D., & Black, A. (2013). The dialog state tracking challenge. In M. Eskenazi, M. Strube, B. Di Eugenio, & J. D. Williams (Eds.), *Proceedings of the SIGDIAL 2013 Conference* (pp. 404–413). Metz, France: Association for Computational Linguistics.

Winograd, T. (1971). Procedures as a representation for data in a computer program for understanding natural language.

Wiseman, S., Shieber, S. M., & Rush, A. M. (2018). Learning neural templates for text generation. *CoRR*, abs/1808.10122.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., Drame, M., Lhoest, Q., & Rush, A. (2020). Transformers: State-of-the-art natural language processing. In Q. Liu & D. Schlangen (Eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (pp. 38–45). Online: Association for Computational Linguistics.

Wong, L., Grand, G., Lew, A. K., Goodman, N. D., Mansinghka, V. K., Andreas, J., & Tenenbaum, J. B. (2023). From word models to world models: Translating from natural language to the probabilistic language of thought.

Xu, W., Deng, Y., Zhang, H., Cai, D., & Lam, W. (2021). Exploiting reasoning chains for multi-hop science question answering. In M.-F. Moens, X. Huang, L. Specia, & S. W.-t. Yih (Eds.), *Findings of the Association for Computational Linguistics: EMNLP 2021* (pp. 1143–1156). Punta Cana, Dominican Republic: Association for Computational Linguistics.

Yadav, V., Bethard, S., & Surdeanu, M. (2019). Quick and (not so) dirty: Unsupervised selection of justification sentences for multi-hop question answering. In K. Inui, J. Jiang, V. Ng, & X. Wan (Eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* (pp. 2578–2589). Hong Kong, China: Association for Computational Linguistics.

Yadav, V., Bethard, S., & Surdeanu, M. (2020). Unsupervised alignment-based iterative evidence retrieval for multi-hop question answering. In D. Jurafsky, J. Chai, N. Schluter, & J. Tetreault (Eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (pp. 4514–4525). Online: Association for Computational Linguistics.

Yang, J., Jimenez, C. E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K., & Press, O. (2024). Swe-agent: Agent computer interfaces enable software engineering language models.

Yang, K., Zhang, T., Cummins, C., Cui, B., Steiner, B., Wang, L., Gonzalez, J. E., Klein, D., & Tian, Y. (2021a). Learning space partitions for path planning. In A. Beygelzimer, Y. Dauphin, P. Liang, & J. W. Vaughan (Eds.), *Advances in Neural Information Processing Systems*.

Yang, S., Zhao, Y., & Tu, K. (2021b). Pcfgs can do better: Inducing probabilistic context-free grammars with many symbols.

Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W., Salakhutdinov, R., & Manning, C. D. (2018). HotpotQA: A dataset for diverse, explainable multi-hop question answering. In E. Riloff, D. Chiang, J. Hockenmaier, & J. Tsujii (Eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing* (pp. 2369–2380). Brussels, Belgium: Association for Computational Linguistics.

Yarats, D. & Lewis, M. (2017). Hierarchical text generation and planning for strategic dialogue. *CoRR*, abs/1712.05846.

Young, S. (2006). Using pomdps for dialog management. In *2006 IEEE Spoken Language Technology Workshop* (pp. 8–13).

Yu, L., Chen, H., Wang, S. I., Artzi, Y., & Lei, T. (2019). Interactive classification by asking informative questions. *CoRR*, abs/1911.03598.

Yu, L., Tan, H., Bansal, M., & Berg, T. L. (2017). A joint speaker-listener-reinforcer model for referring expressions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*.

Yu, S.-Z. (2010). Hidden semi-markov models. *Artificial intelligence*, 174(2), 215–243.

Yu, X. V., Min, S., Zettlemoyer, L., & Hajishirzi, H. (2022). Crepe: Open-domain question answering with false presuppositions. *arXiv preprint arXiv:2211.17257*.

Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent neural network regularization. *CoRR*, abs/1409.2329.

Zhang, H., Zhao, X., & Song, Y. (2020). WinoWhy: A deep diagnosis of essential commonsense knowledge for answering Winograd schema challenge. In D. Jurafsky, J. Chai, N. Schluter, & J. Tetreault (Eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (pp. 5736–5745). Online: Association for Computational Linguistics.

Zhao, C., Xiong, C., Boyd-Graber, J., & Daumé III, H. (2021). Distantly-supervised dense retrieval enables open-domain question answering without evidence annotation. In M.-F. Moens, X. Huang, L. Specia, & S. W.-t. Yih (Eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing* (pp. 9612–9622). Online and Punta Cana, Dominican Republic: Association for Computational Linguistics.

Zhao, W., Chiu, J., Cardie, C., & Rush, A. (2023). Hop, union, generate: Explainable multi-hop reasoning without rationale supervision. In H. Bouamor, J. Pino, & K.

Bali (Eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing* (pp. 16119–16130). Singapore: Association for Computational Linguistics.

Zhou, D., Huang, J., & Schölkopf, B. (2006). Learning with hypergraphs: Clustering, classification, and embedding. *Advances in neural information processing systems*, 19, 1601–1608.

Zhou, W., Hu, J., Zhang, H., Liang, X., Sun, M., Xiong, C., & Tang, J. (2020). Towards interpretable natural language understanding with explanations as latent variables. *Advances in Neural Information Processing Systems*, 33, 6803–6814.

Zhukov, D., Alayrac, J.-B., Cinbis, R. G., Fouhey, D., Laptev, I., & Sivic, J. (2019). Cross-task weakly supervised learning from instructional videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 3537–3545).

Ziegler, Z. M. & Rush, A. M. (2019). Latent normalizing flows for discrete sequences.