Structural Transfer Learning in Semantic Parsers

1060436
MSc in Advanced Computer Science
University of Oxford

Abstract

The widespread success of transformer-based, pre-trained models demonstrates the effectiveness of transfer learning in natural language processing. We perform a fine-grained analysis of transfer learning in the domain of semantic parsing, which is the challenging task of extracting meaning expressed in natural language to precise formal representations including computer code. In doing so, we propose a new methodology for classifying downstream tasks in terms of measurable performance gains from structural inductive biases learned during pre-training. We apply this method to the particular downstream task of converting statements of intent expressed in English to computer code in the Bash programming language, and we show that this task has measurable structural similarities to other machine translation tasks. Finally, we extend this study to other factors contributing to the transferability of structural representations, including compute expended during pre-training and model initialization schemes. We find that English to Bash is strongly similar to a lexical mapping of tokens, and structurally overlaps with English to SQL. We also find that an inductive bias learned from English to German improves the perplexity of an English to Bash semantic parser over a collection of baselines and that imbuing a partial inductive bias from English to German by early stopping during pre-training may match within task transfer in terms of the NLC2CMD metric for semantic accuracy of Bash commands.[a]

[a]Code written for the purposes of this these are included at https://github.com/kyduff/structural-transfer

Contents

Chapter 1

Introduction

Semantic parsers extract precise meaning from natural language. Their aim is to encode semantics expressed in natural language as formal statements in a machine-readable format. Doing so is intrinsically difficult because natural language obeys an ill-specified and implicit grammar that adapts poorly to the precision required by machines.

Machines demand statements that are encoded in strict, unambiguous formats called *formal languages*. To express semantics to a machine, therefore, it is essential to systematically encode meaning in a formal language. Since humans are well-adapted to expressing meaning in natural languages, it would be convenient if machines could parse formal semantic representations from statements expressed in natural language. Deep neural networks are a promising approach to making progress in this domain.

However in practice, automatic semantic parsing is difficult for the following three reasons:

1. Natural language is complex, redundant, and collision prone.
2. Formal languages obey precise syntaxes that must be learned alongside semantics.
3. High quality data are often limited, expensive to collect, or wholly unavailable.

The first and second items are intrinsic to the domain, therefore it is unlikely they will be resolved without substantially perverting the definition of the task. Therefore we will focus primarily on the third item, which is about the nature of our *data*.

Solutions to the semantic parsing problem have been attempted for decades. Before the rise of statistical systems, semantic parsers were mainly constructed by hand, using defining rules based on alignments between grammatical structures in natural language and those in formal languages. Such systems were limited and inherently unscalable. *Statistical methods*, by contrast, capitalize statistical patterns mined from word co-occurrences to implicitly represent the grammatical and semantic structure of natural language. Deep learning models fall into this category of "statistical" methods.

Statistical methods are capable of modeling complex grammatical structures which might escape manually constructed, rule-based models. They are capable of consuming large quantities of data from which grammatical structure can be *inferred*, making them more scalable and robust variations unforeseen by the constructors of manual systems.

Statistical methods have developed alongside the broader fields of neural natural language processing and sequence modeling in recent years. Grefenstette et al. (2014) paved the way for neural networks with a neural semantic parsing architecture adapted from models used in sentiment analysis, document classification, and machine translation. Since then, a dominant approach has been to use *recurrent networks* to model semantic parsing as a sequence modeling task.

A popular recurrent network uses the Long Short-Term Memory (LSTM) unit, which can model linearized versions of complex structures including schema graphs and abstract syntax trees. Its utility is a careful dataflow design that mitigates the problem of exploding and vanishing gradients during training. The prevalence of these models has made it popular to encode semantic parsing as a *sequence transduction* task, whereby an input sequence is transformed into a structured output sequence representing the formal target. This is also a convenient encoding for use in *transformers*, which have recently dominated the literature.

The transformer architecture (Vaswani et al., 2017) is naturally parallel, which makes it ripe for dramatic scaling. The scalability of transformers and broad application space have attracted the attention of several large-scale research labs which have pushed the limits on model capacity and data consumption (Brown et al., 2020; Raffel et al., 2020; Smith et al., 2022). This has lead to enormous, multi-task models that obtain state-of-the-art performance on a variety of tasks, on each outpacing the best performing bespoke models. The performance gain is attributable to *transfer learning*, whereby structural representations learned during pre-training transfer to performance on downstream tasks which fall outside the training distribution.

One example downstream task is the challenge of *program synthesis*. In this task, models must construct a computer program satisfying pre-specified constraints. The Codex model (Chen et al., 2021) was fine-tuned from a large, pre-trained language model for this purpose, and was able to solve over 70% of problems synthesizing programs from their docstring descriptions. More recently, Lu et al. (2022) show that models pre-trained on natural language data can be used to perform more general-purpose computations including bit and list operations like `XOR` and `MIN`.

Despite the staggering performance of transfer learning, very little is understood about it. All of the following are unanswered yet useful questions:

- *Which structural patterns in the training set lead to performance boosts on which tasks?*
- *Can we augment the training data to amplify performance gain or shore up weaknesses in our models?*
- *Are there limits on the effectiveness of unfettered data scaling?*
- *Are pre-trained models always better? If not, when is it better to build a bespoke model?*
- *Is it possible that pre-training on one task may actually* damage *performance on another?*
- *Can we predict the performance change associated with introducing a new form of data?*
- *Can we measure the performance change resulting from pre-training on different kinds of data?*

In this thesis, we focus on the last question. That is, we undertake a careful study of the mechanics of transfer in order to understand its relevance more deeply.

We wish to dissect *which* particular structural patterns mature into downstream performance in semantic parsers. To do so, we require a classification of the semantic parsing task in terms of a structural taxonomy that is determined by measurable performance gains. We measure the gain by fitting structural representations of pre-training tasks into the weights of a neural network. This pre-trained understanding of structure is called a *learned structural inductive bias.* Once the bias of the pre-training task is imbued, we observe the ability of the structural representations to model a semantic parsing task. When applied to a spanning set of pre-training tasks (called *upstream* tasks), the result is a more fine-grained understanding of which structural patterns overlap with the target semantic parsing task. Such an understanding may assist the development of more precise applications of tactical pre-training.

The premise of this study is a hypothesis that some machine translation translation tasks have broader structural overlap with semantic parsing than others. In the context of language modeling, Papadimitriou and Jurafsky (2020) show that structural transfer in LSTMs is correlated with syntactic similarity between languages. This conclusion is unsurprising because we expect linguistic syntax to reflect semantic structure. Why this is interesting is because it indicates that there exists *measurable variation in the strength of transfer between tasks.*

In this thesis we explore the extent of this variation in the context of semantic parsing. We propose a method whereby the structural representations learned on an *upstream* sequence transduction task are evaluated for transferability to a *downstream* sequence transduction task. We then apply this method to the particular downstream semantic parsing task of English to Bash, and compare transferability between several upstreams. Finally, we discuss the implications of our findings and mention some intermediate findings regarding model initialization, evaluation methods, and compute. In total, these findings contribute a method for fine-grained analysis of structural transfer in sequence transduction tasks. Such a method may help clarify our view of transfer learning, which is evidently effective but only poorly understood.

1.1 Outline

- The Background section aims to motivate substantial design choices made in the project by describing available alternatives and justifying the decision made in the context of the hypothesis of the thesis.
- The Method section states the research question of this project, introduces the experimental methodology and motivates its design with reference to other work from the literature.
- The Experiments section defines the upstream datasets used in experiments and motivates their inclusion in the study.
- The Results section states results of the experiment and discusses two further studies on model initialization and compute.
- The Discussion section reflects critically on the methodology and discusses future work and the relevance of findings the to the broader literature.
- The Index of Work contains a list of work completed by the project, complete with clickable links to a description of each included piece of work.
- *To read this thesis with minimal effort, I recommend that you first read the Introduction and then quickly review the Notation. From there you can carry on to the Method and then to the Results, using the Background and Experiments chapters as references for any unfamiliar materials (which will usually be hyperlinked).*

1.2 Achievements

1. We propose a general method for measuring structural transfer between pre-training tasks and a downstream task.
2. We present three artificial sequence transduction tasks and adapt four others for use in the OpenNMT machine translation toolkit.
3. We use the proposed methodology to measure the transerability of structural representations to the downstream semantic parsing task of English to Bash.
4. We report the results and discuss implications of findings, including two further studies on the relevance of compute and model initialization.
5. We provide commentary on the problem of evaluating semantic equivalence of Bash programs and demonstrate several weaknesses in the current standard.

Chapter 2

Background

2.1 Preliminary Materials

Notation

Mathematical basics.

$\mathbf{P}(S)$	denotes the set of probability distributions over a set S.
lg	denotes the logarithm with base 2.
log	denotes the logarithm with base e.
exp	denotes the exponential function with base e.

Definitions.

Linguistic data are usually represented as a corpus of statements which need to be lexically parsed into sequences of "words" (also called "tokens") by a tokenizer before processing by machine learning models. These words are often assigned a non-negative integer index which is used as an input for numerical models, and we identify words with their index. Hence:

Definition 1. A *vocabulary* is a finite set of natural numbers.

Words are joined sequentially into sentences, which are combined to form statements or "utterances." This is a common operation in computer science so we adapt the familiar "Kleene star" notation to represent sets of sentences:

Definition 2. For a vocabulary Σ, define $\Sigma^* := \{(\sigma_1, \ldots, \sigma_n) : n \in \mathbb{N},\ \sigma_i \in \Sigma\}$ to be the set of finite sequences of words from Σ.

Utterances are associated with a language from which some (and usually most) sentences are excluded. *Formal languages* are often determined by a concrete *grammar* which defines the set of sentences in terms of a set of rules and an acceptance procedure by which membership of the language can be tested algorithmically. Natural languages, however, rarely admit precise grammars. Nonetheless, there is still a cultural idea of which sentences are acceptable and which are not. In general, we define a *language* in terms of a prespecified subset of all sentences:

Definition 3. A *language* is a pair (X, Y) where X is a vocabulary and $Y \subseteq X^*$ is a set of sentences.

Remark 1. It is convenient to refer to the pair (X, Y) by X, and overload the symbol X^* to refer to the set of sentences Y. This will be done where the context is unambiguous, otherwise the set of sentences will be referred to explicitly.

We are further interested in modeling a language as it appears in some context, which requires understanding something deeper than the set of sentences. In many cases, it is useful to understand the *distribution* of sentences, beyond a mere understanding of which sentences are acceptable. Thus,

Definition 4. Given a language L, a *language model* is a probability distribution $p : L^* \to [0, 1]$.

In the context of this project, the objects of interest are models which conditionally generate utterances from a *target language* representing the semantic meaning of utterances from a *source language*. It is convenient to represent this as a probability distribution over the set of target sentences. This naturally distinguishes the *decoding strategy* from the probability model. Plus we obtain the further advantage of inheriting analysis methods from the study of language models. Thus we define a *translation model* as follows:

Definition 5. Given two languages, L_1, L_2, a *translation model* is a function $s \mapsto \mathbb{P}(t \mid s)$ where $t_1 \cdots t_n = t \in L_2^*$, and $s \in L_1^*$.

Remark 2. By Bayes's rule, $\mathbb{P}(t_1 \cdots t_n \mid s) = \prod_i \mathbb{P}(t_i \mid s, t_{j<i})$, which gives the basis for sequential sentence generation whereby each target word is generated one-by-one. This will be discussed later.

To produce sentences in the target language using predictions from a translation model we use a *decoding algorithm*:

Definition 6. Given a language L, a *decoding algorithm* is a function $\mathbf{P}(L^*) \to L^*$, which maps a probability distribution over sentences L^* to a sentence in L^*.

Semantic parsing is a particular subfield of translation modeling where the target languages are precisely structured *formal languages*. Here we intend to condition the target distribution based on the *semantics* of the input, in order to produce formal sentences which express the meaning of the source utterance. These formal sentences are ideally machine readable, and can be procedurally checked for validity with relative efficiency. The study of formal languages is rich and it would be distracting to describe it in detail, so we instead adopt the following definition which is good enough to serve the purposes of this project:

Definition 7. A *formal language* is a language $F = (S, \Sigma)$ paired with a *grammar* $G : S^* \to \{0, 1\}$, such that G is efficiently computable, and given a sentence $s \in S^*$, $G(s) = 1$ if and only if $s \in \Sigma$.

Remark 3. In most contexts of computer science, the *grammar* is a set of rules which may be decoupled from the sentence acceptance algorithm. In this project we may use the term "grammar" in the standard sense where the context is unambiguous, however in most cases it is fine to leave the the rule set and the algorithm coupled as in Definition 7.

Hence,

Definition 8. A *semantic parser* is a function $L_1^* \to L_2^*$, where L_1 is a language and L_2 is a formal language.

Remark 4. In this definition alone, we don't assume semantic parsers match semantically aligned sentences. Instead, this is a *property* of a (good) semantic parser, and we aim to optimize this property during training.

While semantic parsers (as defined) are functions which produce utterances, it is convenient to cast them as translation models followed by some (to be specified) *decoding algorithm*. This allows us to consider semantic parsing as a sequence transduction task in the same class as other translation tasks.

A Note on Semantic Parsing as Sequence Transduction

Formal languages are usually symbolic but may not be linear. Some formalisms are *hierarchical*. In this thesis, we consider programming languages, which are usually linear or at least canonically linearizable. It's sometimes common to represent programs as *abstract syntax trees* which are hierarchical. These can be linearized by a traversal algorithm.

The value of linearization is adaptation to models and tasks from the broader domain of sequence transduction. Sequence modeling has a rich ecosystem of translation tasks whose structures may be transferable to the semantic parsing domain. It also broadens the scope of synthetic tasks which may be used to isolate particular syntactic structures of interest. Therefore we find it useful to reduce semantic parsing to a linear sequence transduction task.

Now that the groundwork has been laid, we can discuss some particular features of machine learning which are relevant to this thesis.

2.2 Large Language Models

Large language models (LLMs) are many-parameter language and translation models pre-trained on large corpi of natural language data, usually sourced from the internet. For a sense of scale, the popular LLM GPT-3 has over 170 billion parameters, and consumed over 3000 petaflop-days during training (Brown et al., 2020).

Several large language models, including T5 (Raffel et al., 2020), GPT-3 (Brown et al., 2020), and BERT (Devlin et al., 2019), posted state of the art performance on a variety of tasks. This is interesting because the models themselves were not trained for multiple tasks—in each case, the multi-task performance was emergent from a single training procedure.

This shifts the modeling paradigm from building bespoke models with inductive biases selected appropriately for each task to using expressive model architectures which can capture a broad variety of inductive biases and then *learn* the appropriate inductive biases from large quantities of (potentially noisy) data.

The multi-task performance of these large models is potentially attributable to transfer learning, and their performance indicates a large amount of homogeneity across language tasks. A systematic study of transfer learning could refine our understanding of this mechanic, which could enhance modeling techniques for contexts where massive language models are unavailable or impractical.

2.3 Description of Task

Semantic parsing broadly includes all translation tasks where source semantics are encoded in natural language and the target representation is a formal language.

In this thesis, we focus specifically on semantic parsers which target computer code as their formal language. In this setting, input statements describing the intent of a program in natural language are paired with formal implementation of the desired program in computer code.

If a precise specification of inputs and desired outputs or constraints is also included in the description, the task is referred to as *program synthesis*. Program synthesis is intrinsically difficult because because the space of programs is massive. For any given statement of intent (or program specifications) there is an infinite collection of potential programs and no natural way to search it. There is no natural way to search this collection because rewards are *sparse*, meaning that a correct program is known only once it is found, and there is no sensible way to measure progressive incremental correctness.

In our case, we assume the additional challenge of synthesizing programs which implement intentions specified in natural language, and with no way to verify that the program is correct.

In this case, we encounter these additional challenges:

- Ambiguities in natural language are difficult to resolve decisively
- Structural patterns in natural language are implicit and often hidden

Additional injections of data are often sufficient to resolve these issues, however doing so is not always available. For cases of low-resource source or target languages, additional data may be unavailable, so a popular approach is to *transfer* performance from a high-resource domain to low-resource domains. In this thesis, we study the mechanics of such transfers.

Flavors of Semantic Parsers

We are concerned only with semantic parsing tasks whose targets are programming languages. Within this context there are two flavors of tasks:

1. Contextual
2. Non-contextual

In *contextual* semantic parsing task, an additional context is included with the natural language inputs. One such example is the natural language to SQL task, in which the table scheme is included with natural language inputs, and must be considered by the model to generate appropriate queries. This task is used in this thesis, and a more detailed description of SQL and its relevance to this project will be discussed later. Otherwise the context may be code extracted from a codebase which informs which program ought to be imputed or which programming styles should be adopted. This is the case with the popular CONCODE task (Iyer et al., 2018), where models are challenged to complete implementations of Java classes based on docstring descriptions.

Alternatively, semantic parsing tasks may be *non-contextual*. In this case, only the description of intent in natural language is included as input, and the model is tasked to synthesize the appropriate program based on this description alone. This task is arguably simpler than the contextual one, and for that reason this project has opted to focus on a non-contextual target. Popular datasets for English to Python including CoNaLa (Yin et al., 2018) are of this type and have a rich literature surrounding them. Also of the non-contextual type is the English to Bash task proposed by Lin et al. (2018). Bash has the additional advantage of relatively short target sentences and a comparatively simple application space which consists largely of chained utilities which operate on text data. This makes Bash a ripe target for study in this project.

2.3.1 Bash

Bash[1] is a shell scripting language commonly used in Unix-like operating systems including Linux and macOS. It is a Turing complete programming language which supports functions and integer arithmetic. The only datatype in Bash is string, and it was designed for maximal interoperability and ease-of-use.

Bash is most frequently used to interact with the operating system kernel and perform operations on a machine's filesystem. Programs usually consist of concise "one-liners" that perform some action and return a result in the form of text printed to the console, also called `stdout`. The output of a command can be redirected, and can even be used as the input to another Bash command. Due to this property, Bash commands are naturally composed with one another into a chain of commands called a *pipeline*, which is instantiated by the character "`|`" (pronounced "pipe").

Pipelines

In each step of a pipeline, the output of one command is fed as the input to the next. As an example, the Bash command `cat file.txt` reads the content of the file "`file.txt`" to `stdout`. The command `wc -l` counts the number of lines in its input.[2] In this case, `wc` is the *utility* and stands for "word count," and the modifier `-l` instructs `wc` to count lines instead of words. This modifier is called a *flag*. Flags are denoted with the symbol "`-`" or "`--`". Now, to combine `cat` and `wc` into a pipeline we write

```
cat file.txt | wc -l
```

which evaluates to the number of lines in `file.txt`.

Subshells

Additionally, Bash supports composition in another fashion called a *subshell*. Subshells spawn a new interpreter session and execute the contained bash program in this new session. A subshell is created by enclosing a program in "`$( )`", and the output of the enclosed program replaces the subshell in the invoking sentence. For example, the command `basename` prints the name of a directory excluding any parent directories, and `pwd` prints the current working directory. Hence, the command

```
basename $(pwd)
```

[1] https://www.gnu.org/software/bash/

[2] In reality, there is a subtle difference between input to a utility as a file or as an argument. This difference is largely irrelevant to this project so we conflate the two notions of input.

prints the name of the current working directory excluding any parent directories.

With these two operations, Bash contains compositional structure which is both sequential (pipelines) and hierarchical (subshells). This alone makes Bash an interesting target to study. With the additional convenience of short, self-contained, and easily describable programs, Bash is a natural choice for our task of study in this project.

2.4 Evaluation

2.4.1 Challenges

Because Bash is Turing complete, determining the equivalence of programs is undecidable. Therefore script equivalence must be approximated by more practical evaluation methods. There are three popular approaches:

1. Exact match accuracy
2. Heuristic accuracy
3. Execution accuracy

Exact match accuracy is measured by either by comparing the character strings (or alternatively token sequences) composing a program. Literally identical programs are obviously equivalent, so this approach provides a lower bound on a model's accuracy. However, programming languages frequently support several implementations of the same semantics, and two algorithms may produce the same result though their syntax is different. It is also possible for programming languages to be invariant to some textual manipulations. For example, in Bash commands, the *order* of flags is usually irrelevant to the semantic meaning of the command (though their *presence* is important!).

Heuristic-based systems intend to resolve these problems by normalizing for known equivalences within a given programming language. They usually work by reducing programs to a normalized form and measuring properties about the program in normal form. These properties can be aligned between programs to measure equivalence, and may also be used to construct and idea of *distance* between programs. The NLC2CMD metric for Bash used in this thesis is a metric of this type.

Heuristic systems are usually incomplete and always fail to capture every potential program variation. They also fail to resolve differences which may be context dependent, since heuristics are computed in isolation from any context. Execution accuracy is an attempt to resolve this problem with parallel execution is a controlled environment. Specifically, two programs are executed in a controlled environment and their outputs are compared. The same output from both programs is evidence the programs are equivalent. With execution accuracy we cannot escape the problem of *spurious equivalence*, whereby two programs may have identical outputs in the tested contexts though there may exist other contexts in which the outputs differ. This problem can be mitigated by executing programs in several (ideally random) contexts, though this is usually difficult to orchestrate without substantial effort and care.

In general, execution accuracy is almost always preferable to the other two methods though it is more difficult to compute. Unfortunately, a stable system for evaluating execution accuracy is absent from the English to Bash literature, so in this project we resort to a heuristic-based method and cross-reference the result with other evaluation methods taken from the language modeling domain.

2.4.2 Methods

We apply three methods in total.

Perplexity

A language model L is a probability distribution over the set of sentences constructed from a vocabulary T. A popular and vocabulary agnostic metric for performance of language models is the *perplexity*, which is formally defined as

$$\mathrm{ppl}(X) = 2^{H(X)},$$

where X is a random variable distributed by L. Given a set of samples $t_1, \ldots, t_n \in T^*$, the perplexity of L can be estimated by

$$\mathrm{ppl}(L) = 2^{-\frac{1}{n}\sum_i \lg(L(t_i))}.$$

Given a set of samples $(s_1, t_1), \ldots, (s_n, t_n) \in S^* \times T^*$, we can evaluate the performance of a machine translation model by viewing it as a map $L : s, t \mapsto \mathbb{P}(t \mid s)$ and computing

$$\mathrm{ppl}(L) = 2^{-\frac{1}{n}\sum_i \lg(L(s,t))}.$$

Accuracy

Alternatively, given a target label, you can measure the "accuracy" of a language model as a heuristic for its effectiveness. Given target sentence $t = t_1 \cdots t_n$ (some of whose tokens may be a special "`<pad>`" $=: \square$ token), we can evaluate the *accuracy* of a predicted sentence $t' = t'_1 \cdots t'_n$ as follows:

$$\mathrm{acc}(t' \mid t) = \frac{\sum_i \mathbb{1}(t_i = t'_i)\mathbb{1}(t_i \neq \square)}{\sum_i \mathbb{1}(t_i \neq \square)}$$

In plain English, this parses as the number of tokens in the predicted sentence which are equal to their corresponding token in the target sentence if that token is not padding. This quantity is then divided by the length of the target sentence (subtracting padding) to give a normalized figure which can be rendered as a percentage.

To measure the accuracy of an entire corpus of sentences, we take the total number of correct tokens across the whole corpus and divide by the total number of tokens across all target sentences. That is, for a corpus of target sentences $T = \{t^j : j = 1, \ldots, N\}$ and a parallel corpus of predictions $P = \{p^j : j = 1, \ldots, N\}$, we can lift the definition of acc to the corpus pair by

$$\mathrm{acc}(P \mid C) = \frac{\sum_j \sum_i \mathbb{1}(t_i^j = p_i^j)\mathbb{1}(t_i^j \neq \square)}{\sum_{j,i} \mathbb{1}(t_i^j \neq \square)}$$

This is neatly recorded in an online fashion on one forward pass of a parallel corpus by keeping a running tally of the number of correct parallel (non-padding) tokens and a running tally of the total number of non-padding tokens in the target corpus. To simplify the computation, we use a greedy decoding algorithm to generate sentences from predicted distributions during training and validation loops.

NLC2CMD

The NLC2CMD paper (Agarwal et al., 2021) proposed a heuristic for program similarity evaluation to be used in the NLC2CMD competition held at the NeurIPS conference in 2020. For the purposes of the competition, the metric was designed to be an easily computable, quantitative measure of program similarity that simultaneously rewards utility correctness while penalizing excess flag predictions.

It is computed as follows. Given a ground truth Bash command C_g and a predicted command C_p in the form of a pipeline of utilities, let $U_g^1, \dots, U_g^T$ be the ground truth utilities in order, where T is the maximum number of utilities between the prediction and the ground truth. Define $U_p^1, \dots, U_p^T$ similarly. For a given utility U of a command C, let $F(U)$ be the set of flags passed to U in C. Given U_g^i and U_p^i, let $N = \max(|F(U_p^i)|, |F(U_g^i)|)$ be the maximum number of flags between the two. We then define the *flag score* as

$$S_F^i(C_p, C_g) = \frac{1}{N}\left(2|F(U_p^i) \cap F(U_g^i)| - |F(U_p^i) \cup F(U_g^i)|\right)$$

Intuitively, this is a normalized heuristic that rewards correct flag predictions and penalizes excess flag predictions in a way that is agnostic to flag ordering. This is important, because flag order is nearly always irrelevant in Bash. Now, we can compute the *prediction score* as

$$\text{NLC2CMD}(C_p, C_g) = \sum_{i=1}^{T} \frac{1}{T}\left(\left[\mathbb{1}(U_P^i = U_g^i)(1 + S_F^i)\frac{1}{2}\right] - \mathbb{1}(U_p^i \neq U_g^i)\right).$$

The result is a heuristic in the range of -1.0 (worst) to 1.0 (best) that rewards sequentially matching utilities in a way that is scaled by flag score. Tran et al. (2019) show that BLEU score is poorly correlated with program semantics, so we opt to omit BLEU score and consider NLC2CMD as a metric for semantic similarity of Bash programs. However, this project discovered substantial weaknesses in this metric which will be discussed in a later section.

2.5 Modeling

2.5.1 Architecture

Reccurrent Networks

Recurrent networks are a sequence modeling technique derived from the study of dynamical systems whereby a *hidden state* is updated with progressive observations from a sequence (Goodfellow et al., 2016). The idea is to construct a recursive *computation graph*, with learnable parameters that propagate information to progressive steps of the computation. For example, the following relationship defines a recursive computation graph that is the simplest non-trivial recurrent network

Let $h_0 \in \mathbb{R}^h$ be an initial state and let $x_1, \dots, x_n \in \mathbb{R}^h$ be given. We predict $\hat{y}_1, \dots, \hat{y}_n$. For each t, we compute

$$\begin{aligned} a_t &= b + Wh_{t-1} + Ux_t \\ h_t &= \sigma(a_t) \\ o_t &= c + Vh_t \\ \hat{y}_t &= \text{softmax}(o_t), \end{aligned} \tag{2.1}$$

where W, U, V are learnable projection matrices, c and b are learnable bias vectors, and σ is a non-linear activate function.

To train the recurrent network, we evaluate each $\hat{y}_i$ with a loss function and compute the gradient with respect to each parameter in W, U, V, c, b. The gradient at the i-th step depends on the loss function computed at the j-th step for all $j > i$, so we must retain the trace of the entire computation graph during training. This is called *unrolling* or *unfolding* the network.

There are two major obstacles to training recurrent networks:

1. The network must be computed sequentially and the entire network must be unrolled (to potentially variable sizes) in order to accurately compute gradients.
2. Due to the chain rule, gradients in the early stages of computation are scaled by gradients in later stages.

The second obstacle may cause gradients at earlier stages of the computation graph to either grow extremely large or shrink to zero if the gradients are not carefully controlled. In the former case this is called the *exploding gradients problem* and in the latter case this is called the *vanishing gradients problem*.

To resolve this problem, more complicated recurrent units than Network 2.1 have been devised. A popular alternative is the LSTM[3], which includes an additional channel that allows information flow to earlier steps of the computation in a weighted fashion that may encourage long range data flow or otherwise prevent it. This stabilizes training though LSTMs still suffer from sequential computations that prohibit scaling. This has made the transformer a popular alternative for sequence modeling.

Transformer

The *transformer* (Vaswani et al., 2017) is a non-recurrent sequence modeling architecture that uses *multi-headed self attention* to condition a tokenwise probability distribution on the content of an input sequence. Each token vector is replaced by a *weighted average* of all token values in the sequence. When applied repeatedly, the result is a progressive mixing of information.

To be more precise, the mechanism works as follows: a *query* matrix $Q \in \mathbb{R}^{N \times d_k}$, *keys* matrix $K \in \mathbb{R}^{N \times d_k}$, and a *values* matrix $V \in \mathbb{R}^{N \times d_v}$ are are obtained by projecting N embedding vectors to the relevant space using learnable linear projections W_Q, W_K, and W_V.

The queries and keys are combined with a standard Euclidean dot product, and scaled by $\sqrt{d_k}$ to stabilize gradients. The result is converted into a probability distribution using the *softmax* operation. Softmax converts a vector into a probability distribution by element-wise exponentiation followed by normalization:

$$\text{softmax}(v)^i = \frac{\exp(v^i)}{\sum_j \exp(v^j)}.$$

When applied to a matrix, we compute the softmax row-wise.

Putting this together formally, we compute the "attention" of Q, K, V as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V,$$

[3]Short for Long Short Term Memory

By using different projection matrices W_Q, W_K, and W_V, we obtain different attention mixtures of V. This is called "multi-headed" attention.

This is relevant for a two reasons. First, the bulk of the operation is matrix multiplication which can be vectorized and efficiently computed on a GPU. Since very few sequential operations are required, data can be batched which leads to large data throughput. Furthermore, each attention head can be computed independently, and even distributed across several devices. This enables the dramatic model scaling used to train GPT-3 with 170 billion parameters (Brown et al., 2020) and the so-called "Megatraon-Turing NLG" model with a staggering 530 billion parameters (Smith et al., 2022). Second, multi-headed self attention is not inherently sequential insofar as information does not necessarily flow from the beginning of a sequence to the end.

Consequently, transformers do not have intrinsic sequential bias at the architecture, and any sequential bias must be *learned*. Contrast this with reccurent models which incrementally update a hidden state at each token: at the n-th token, the hidden state has been exposed only to representations generated by the preceeding tokens. This imbues a sequential inductive bias in recurrent models that may be useful for modeling natural and formal languages, which is often sequential. Because of this, we expect any sequential bias learned by transformers during upstream training will improve performance on the downstream semantic parsing task.

Remark on Relevance to the Thesis

Transformers have demonstrated a capability for high performance on a broad variety of tasks (Raffel et al., 2020). This empirically supports the hypothesis that the transformer architecture is capable of learning the inductive biases associated with a diverse range of sequence transduction tasks. In our case, we require a model expressive enough to capture all the structural patterns in the palette of upstream datasets used in the project. Transformers are the industry standard for modeling all of the upstream datasets of interest to this thesis, so it is a natural choice for our model.

Embeddings

In PyTorch, an embedding layer from vocabulary V to a hidden dimension h is implemented as a matrix $W \in \mathbb{R}^{|V| \times h}$.

This serves as a simple lookup table whereby $i \in V$ is embedded as W^i, which is an h dimensional real vector. Since transformer attention heads do not have intrinsic sequential bias, the embedding layer is usually followed by a positional encoder which mixes the embedding vectors with a fixed sinusoid whose frequencies encode the position of each token in the sequence. In our case, we augment an input matrix $I \in \mathbb{R}^{N \times h}$ by

$$I' = I + P$$

where

$$P_i^j = \begin{cases} \sin(j/10000^{i/h}) & i \text{ even} \\ \cos(j/10000^{(i-1)/h}) & i \text{ odd} \end{cases}$$

This allows the model to distinguish between the same token t appearing in different positions in different sentences.

The embedding matrix W is initialized randomly and trained according to a specified optimization strategy. For gradient-based methods, this usually entails updating each parameter with respect to a loss function of W.

Training embeddings has the ability to align tokens to representations present in deeper model layers. Frequently, embeddings are pre-trained and frozen for the model to adapt to (Pennington et al., 2014; Mikolov et al., 2013). In our case, the *model* is frozen and the embeddings are learned. Crucially, an embedding layer is equivalent to a linear projection, which is simply a matrix multiplication. This means embeddings are incapable of learning any higher order structural behavior.

2.5.2 Training

Cross Entropy Loss

Compute cross entropy between the predicted distribution and a distribution induced by the target sentence (in the un-regularized case, we compare to the Kronecker distribution) Let T be a vocabulary and suppose the maximum sentence length is N. Given two probability distributions $p, q : T \to [0, 1]$, the *cross-entropy of p relative to q* is computed by:

$$\text{xent}(p, q) := -\sum_{t \in T} \log(p(t))q(t) \tag{2.2}$$

Since a translation model predicts a probability distribution over sentences, we can't compare directly to the target sentence, we need to compare to a *distribution.* The simplest such distribution is the Kronecker distribution, which assigns a probability of 1 to each target token in the sequence, and 0 otherwise. This means given a target sentence $t = t_1 \cdots t_n$,

$$\mathbb{P}(t'_j \mid t_{i<j}) = \mathbb{1}(t'_j = t_j)$$

This is easily encoded as a tensor $\{0, 1\}^{|T| \times N}$ where N is the maximum sentence length. Given a training sample s, t and a target word as t_j, denote the Kronecker distribution on t_j as $\delta(t_j)$ and the model's predicted distribution by $P(\cdot \mid s, t)$. If we combine with equation 2.2, we obtain

$$\text{xent}(P, \delta(t_j)) = -\log(P(t_j \mid s, t)).$$

In most cases of machine translation, we use this distribution as the target. However we may also use a smoothened version of this, for example as in label smoothing.

Cross-entropy loss has the additional benefit of differentiability, which makes it amenable to gradient based optimization methods, which are used ubiquitously in machine learning. Glorot and Bengio (2010) demonstrate the sharpness of a cross-entropy loss surface compared to a quadratic loss surface which may improve convergence rate by increasing gradient norms. Essentially, this makes for more informative gradients.

2.5.3 Initialization

Uniform Initialization

Uniform initialization is a parameterized method for randomly initializing weights in neural networks. Given a hyperparameter α, the weights of a given tensor are initialized with a sample from $\text{Unif}[-\alpha, \alpha]$, the uniform distribution on the interval $[-\alpha, \alpha]$.

This is a simple technique that is frequently used in neural networks. However, it may subject the gradients to behavior that is inconsistent with depth (Glorot and Bengio, 2010) and introduces the hyperparameter α, which must be tuned.

Xavier Initialization

Xavier initialization was defined by Glorot and Bengio (2010) after an observation that traditionally initialized networks have activation and gradient statistics that are not stationary with model depth. Where n_i is the input dimension and n_o is the output dimension of a tensor W, Xavier initialization randomly samples the values of W according to a uniform distribution $\text{Unif}[-a, a]$ where

$$a := \sqrt{\frac{6}{n_i + n_0}}$$

2.5.4 Regularization

Dropout

Dropout is a regularization technique that approximates ensembling thinned neural networks by randomly dropping feedforward values during training (Srivastava et al., 2014). The idea is to randomly omit a given weight with some probability p to avoid excessive influence of any single weight. In practice, this is implemented by prepending to each network layer a *dropout filter* which sets each element of the input to 0 with probability p. At evaluation, the dropout filters are removed and the outputs are scaled by $1 - p$.[4] This ensures the resulting model is equivalent in expectation to an ensemble of thinned networks which have been independently trained (Srivastava et al., 2014).

In this project, dropout is used to regularize all models during pre-training. This stabilizes training and improves generalization of learned structural representations.

Label Smoothing

It is nearly impossible to predict a Kronecker distribution in practice, which makes this an unfair target. As a regularization technique, we can instead contrast the predicted softmax distribution with a smoothed Kronecker distribution, which models some amount of noise in the labels. This is called *label smoothing*. In particular, we can introduce a hyperparameter ε, and introduce noise by setting the target classification to have probability $1 - \varepsilon$, and the $|T| - 1$ other classes to have probability $\varepsilon/(|T| - 1)$ (Szegedy et al., 2016). This formalizes a model for noise wherein a label is incorrect with probability ε with a uniform prior on the alternatively correct label.

2.5.5 Optimization

Stochastic Gradient Descent

Gradient descent is a popular optimization algorithm which exploits the property that gradients point in the direction of steepest ascent. Computing the gradient is computationally

[4]Equivalently, the outputs may be scaled by $1/(1 - p)$ during training. This is the implementation used in PyTorch.

infeasible in most settings, so the gradient is approximated stochastically on a *mini-batch* of training samples. Weights are updated after each mini-batch according to

$$w \leftarrow w - \frac{\eta}{B} \sum_i \nabla f_i(w), \tag{2.3}$$

where $f_i(w)$ is the loss function evaluated at the i-th data point in the mini-batch.

The parameter η is a hyperparameter called the *learning rate*. It is often tuned scrupulously and decayed either in response to performance statistics on a held-out validation set or according to a schedule. In the standard version of stochastic gradient descent described above, gradient estimates can be unstable and the learning rate is uniform across all weights. This can lead to erratic training and suboptimal results.

Adam

Adam (Kingma and Ba, 2015), by contrast, replaces the update rule 2.3 with an adaptive update rule that statefully maintains estimates of the first and second moments of the stochastic gradient distribution. For each parameter, an exponentially weighted moving average of the first and second moment are computed online at each step t according to

$$\begin{aligned} m_t &\leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &\leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2, \end{aligned}$$

where β_1, β_2 are hyperparameters, g_t is the estimated gradient, and g_t^2 is the element-wise square of g_t.

This technique of replacing update steps with a linear combination of previous values is called *momentum*. This biases steps to be similar to recent steps, and can denoise updates. The first and second moment estimates are normalized according to $\hat{m}_t = m_t/(1-\beta_1^t)$ and $\hat{v}_t = v_t/(1-\beta_2^t)$ (here β_i^t refers to β_i raised the t-th power), and the weights w_t are updated according to

$$w_t \leftarrow w_{t-1} - \eta \cdot \hat{m}_t/(\sqrt{\hat{v}} + \epsilon),$$

where $\epsilon > 0$ is some small perturbation introduced for numerical stability.

Thus, the algorithm adaptively updates each weight element based on an exponentially weighted moving average of the mean and variance of the gradient distribution. The updates are proportional the *ratio* of mean to variance, which the authors call the “signal to noise ratio” (Kingma and Ba, 2015). Intuitively, $m \ll \sqrt{v}$ indicates high uncertainty about the direction of the true gradients, and Adam elects to take smaller steps in this case.

Adam has two tunable hyperparameters besides the learning rate: β_1 and β_2. Interestingly, these parameters rarely require extensive tuning and off-the-shelf values typically produce satisfactory results (Kingma and Ba, 2015). Because of this, stock values were assumed for these parameters based on other transformer models from the machine translation and semantic parsing domains (Fu et al., 2021).

2.5.6 Decoding

Translation models as defined in this thesis are maps $s \mapsto \mathbb{P}(t \mid s)$ from sentences of a source language to a probability distribution over sentences of a target language. Given such a probability distribution, we are tasked with producing an utterance from the target language satisfying an objective criterion. In machine translation, the objective is to produce an utterance in the target language with the same semantic meaning as the input utterance.

The space of sentences for most languages is prohibitively large, so it is inconvenient and often intractable to sample directly from the distribution over sentences. Fortunately, Bayes's Rule lends naturally to a sequential algorithm which is well-suited to most machine translation model implementations. In particular, we can decompose a sentence t into constituent words $t_1, \ldots, t_n$ where $t = t_1 \cdots t_n$. Then, by Bayes's Rule,

$$\mathbb{P}(t \mid s) = \mathbb{P}((t_1, \ldots, t_n) \mid s) = \prod_i \mathbb{P}(t_i \mid s, t_{j<i})$$

From here it is natural to construct the target utterance by first setting t_1 according to some function of $\mathbb{P}(\cdot \mid s)$, then t_2 according to some function of $\mathbb{P}(\cdot \mid s, t_1)$, and so on.

Greedy Search

A simple way to generate each token t_i is to sample it directly from the distribution $\mathbb{P}(\cdot \mid s, t_{j<i})$. This is convenient, though it is non-deterministic and may result in ill-formed decoded utterances.

Alternatively, we can pick t_i *greedily* whereby $t_i := \operatorname{argmax} \mathbb{P}(\cdot \mid s, t_{j<i})$. This approach has the benefit of computational efficiency: only one distribution needs to be held in memory and only a linear scan over the distribution is required to obtain the desired value.

Greedy decoding suffers from the drawback of local optimization. This may cause the decoding algorithm to select some $t_i = \operatorname{argmax} \mathbb{P}(\cdot \mid s, t_{j<i})$ when there may exist another $t_i' \neq t_i$ for which the aggregate sentence is more probable, that is

$$\mathbb{P}(t_1 \cdots t_i \cdots t_n \mid s) < \mathbb{P}(t_1 \cdots t_i' \cdots t_n \mid s).$$

To prevent this, we may introduce a global lookahead algorithm whereby *every* branch from t_i is explored. We'll call this *global search.*

Global Search

By searching every possible branch, global search avoids locally optimal tokens that reduce the quality of ultimately accepted sentences. While theoretically optimal, this algorithm is exponential in sentence length and therefore usually intractable and eliminates the benefit of sequential generation.

As a middle ground, we approximate exhaustive search by bounding the memory usage of the algorithm with a *beam width*, say k, which explores only the top k branches. This is called *beam search.*

Beam Search

At each step of decoding with beam search, the conditional distribution of the next token is computed over the entire vocabulary. Only the k most likely partial decodings are retained, and the conditional distribution of the following token is computed for each of the k partial decodings. Out of *these*, the top k partial decodings are retained and the others are discarded. The algorithm continues until the sequence is complete.

Because these k partial decodings are the "best" candidates in terms of likelihood, beam search is sometimes called (a variant of) *best first* search. Using this algorithm we reduce the risk of suboptimally decoding greedily without explosive memory usage. In this thesis, we use beam search with a beam width of 5 for our decoding algorithm of choice.

2.5.7 Toolkit

OpenNMT

OpenNMT (Klein et al., 2017) is an open-source toolkit for machine translation with implementations in PyTorch and Tensorflow. Bundled in the toolkit are stable implementations of a broad variety of machine translation components, including bidirectional recurrent encoders, pyramidal encoders, convolutional encoders, LSTMs, and Transformers. Besides this, the toolkit supports dynamic batching, sequence padding, masking, and data parallelism out-of-the-box. These are common operations in machine translation which can stabilize training. All of the above features are standardized and exposed over a YAML[5] interface which makes it easy to serialize hyperparameters in a format that is simultaneously human and machine readable.

Alternatives to OpenNMT include Fairseq (Ott et al., 2019) and Trax[6] which both provide an extensive library of sequence modeling tools. While suitably expressive to implement the models studied in this thesis, these toolkits are considerably more complex than OpenNMT.

Some custom adjustments were made by the thesis to accommodate the particular specifications required by some experiments. These include freezing the encoder and decoder weights, support for weight decay in the Adam optimizer, and detailed experiment tracking using the Weights and Biases platform,[7] among others. Some features were contributed to the codebase in the form of a pull request. The feature implementing encoder and decoder freezing was contributed to the OpenNMT repository in the form of a pull request.[8]

The available features make it easy to rapidly experiment with different configurations with minimal developer overhead. The maturity of the platform allows for elegant serialization of experimental parameters as YAML files which is replicable. The open-source implementation has been reviewed by hundreds of contributors from prestigious institutions, which reduces the chances of an implementation bug that may invalidate results.

[5]https://yaml.org/
[6]https://github.com/google/trax
[7]https://wandb.ai/site
[8]https://github.com/OpenNMT/OpenNMT-py/pull/2176

Chapter 3

Method

3.1 Research Question

The aim of this project is to develop a refined understanding of the mechanics of structural transfer in semantic parsers. Such an understanding would improve methods for classifying downstream semantic parsing tasks according to structural features, which may lead to more efficient applications of transfer learning.

Structural transfer is understood by this project as a *learned inductive bias* in the form of *structural representations* by the weights of a transformer. This means that a transformer learns a weight surface that encodes the structure of a pre-training task, which, when weights are frozen, changes the inductive bias of the transformer. This structural inductive bias may be relevant to a target task other than the pre-training task if that target task shares structural patterns with the pre-training task.

To understand the structural taxonomy of tasks more deeply, it is useful to understand *which* structures lend to effective transfer between tasks. This would clarify which *particular* elements of a pre-training strategy produce gains in the target task and therefore expose structural intersections between tasks which may be exploited to produce more efficient pre-training schemes for special purpose models.

In the case of this thesis, we study the downstream target of *semantic parsing*. Our aim is to ascertain which structural patterns imbue inductive biases which are effective for semantic parsing. To attain empirical validity, we seek a method for measuring structural transfer between an isolated pre-training task and our downstream target. It is important that the structural overlap is reliably *measurable*, because this provides the means of evaluating comparative relevance across a span of tasks, which makes any result far more interesting. To this end we propose a methodology to experimentally determine the structural qualities of sequence transduction tasks in general, and then apply it to the particular case of semantic parsing.

This leads us to pose the research question: *which structural inductive biases measurably transfer to semantic parsing?*

3.2 Related Work

Sinha et al. (2021) run a similar experiment on an encoder-only transformer with masking.

This experiment is particularly interesting because they demonstrate a robustness of pre-training to random reordering in sentence data. This suggests that word co-occurrence statistics may be enough to improve language modeling performance on downstream tasks.

Lu et al. (2022) take a different approach and fix the pre-training dataset and test zero-shot performance on set of natural and synthetic downstream sequence transduction tasks with minimal fine-tuning. This is an interesting line of study, though subtly different from the hypothesis of our project. In this thesis, we aim to classify the structural properties of the *downstream* task, whereas Lu et al. (2022) instead determine the generalization of structural patterns in the *upstream* task.

Papadimitriou and Jurafsky (2020) set up an experiment for language models whereby LSTMs are pre-trained on a so-called "L1" dataset, frozen, and then evaluated on a downstream,[1] task. We take inspiration from Papadimitriou and Jurafsky (2020), who apply their method to the downstream task of language modeling. In particular, modeling the Spanish language with an LSTM. Papadimitriou and Jurafsky (2020) thoughtfully control for vocabulary overlap, token distribution, corpus size, sentence length, compute, and model shape. We attempt to adapt this to the sequence transduction domain.

Discussion on Differences in th Language Modeling Domain

In this thesis, we study structural transfer from *translation modeling* tasks rather than language modeling tasks.

This is interesting because translation models are associated with a language *pair* rather than a single language. In particular, sequence transduction tasks are encoded in the form of bitext sentence pairings which induce an implicit relationship between source and target sentences. This relationship will be called the *translation rule*. Such a translation rule must be defined for each pre-training task and constrains the set of sentences available in the target language once the source language is given, and vice versa. This makes it more difficult to simultaneously control for the distributional features of *both* languages without making an arbitrary, synthetic, or non-generalizing translation rule.

That is, given a source sentence distribution (with features like sentence length and token distribution), a deterministic translation rule induces a distribution over the target sentences. In order to control for the distributions of both the source and target sentences simultaneously, you need to pick your translation rule from the set of functions which reflect and preserve the desired distributional features. This is *hard*.

Furthermore, not only do the *distributional* features need to generalize, but so does the *translation rule*. To formalize this, we use the following notation.

In the language modeling task, there is only one language and therefore no translation rule. You are given a language X, and the task is to learn a language model over the set of sentences X^*.

In the translation modeling case you are given a source language S, a target language T, and a set of pairings $f \in S^* \times T^*$. These pairings f are considered the *translation rule*. In many cases, f can be conceptualized of as a function $S^* \to T^*$.[2]

This introduces three levels of task specific structure:

[1] In their lexicon, this is called an "L2" task

[2] In reality f is *not* a function because it is possible that a sentence s may correspond to several (equivalent) target sentences $t_1, \ldots, t_k$. As a simple example, the sentence "a formula for adding 3 to 4" may parse equally well to "4+3" or "3+4".

1. Source language
2. Target language
3. Translation rule

Once S^*, T^*, and f are given, a distribution $\Pi \in \mathbf{P}(S^*)$ determines the distribution on T^*, since for any $P \in \mathbf{P}(T^*)$, $P(t)$ must be equal to $\Pi(f^{-1}(t))$ for all $t \in T^*$.[3]

This makes it clear that within a task (S^*, T^*, f), we must be careful to study the structural features of S^* and T^*, as well as understand the implications of the choice of f in the case of synthetic tasks.

Defining the task in this way and studying a rich variety of tasks with appropriate baselines can generalize techniques from the language modeling literature to the sequence transduction domain. Furthermore, we include baselines of randomly initialized models which aren't pre-trained. Papadimitriou and Jurafsky (2020) omit such baselines from their study. However findings from this thesis and work by Wieting and Kiela (2019) demonstrate that doing so is critical.

3.3 Hypotheses

The central hypothesis of this thesis is that some pre-training tasks are more relevant than others in semantic parsing. We model this difference in terms of *structural transfer* whereby the inductive bias learned from one task is transferred to semantic parsing. This reflects the underlying idea that the "relevance" of a pre-training task is attributable to structural similarities with the downstream target task.

Linguistic structure is typically reflected in syntax, so it's likely that syntactically similar language tasks will have strong structural overlap. Papadimitriou and Jurafsky (2020) show that this is the case in natural language models, and we expect the findings to generalize to semantic parsing. Thus, we expect upstream datasets sampled from within the semantic parsing domain will transfer effectively to the downstream task when compared to synthetic or natural machine translation tasks which are less substantially related to the downstream task. Regardless of the result, it would be valuable to observe variation in upstream task relevance since this would precede any effort to exploit these variations to create more efficient pre-training schemes.

3.4 Model Architecture

- The model consists of an embeddings layer followed by a standard transformer as described by Vaswani et al. (2017).
- Specifically, the model consists of a transformer encoder-decoder stack with 6 layers, each with 8 self-attention heads.
- The decoder layers are autoregressively masked, the feedforward layers have dimension 2048 and the hidden size is 512.
- A graphical representation of the model can be found in Figure B.1
- All models are regularized with dropout and label smoothing rates of 0.1, which is consistent with standard models in the field (Vaswani et al., 2017; Fu et al., 2021).

[3]Here $f^{-1}(t)$ denotes $\{s \in S^* : (s, t) \in f\}$.

- All learning rates follow the so-called "noam" schedule originally proposed by Vaswani et al. (2017), which consists of a linear warm-up phase followed by inverse quadratic decay.
- A batch size of 2048 tokens was used. Dynamic batching was implemented to avoid out-of-memory errors and maximize memory utilization.
- All models are trained to convergence on two NVIDIA GeForce GTX 1080 Ti GPUs, each with 11GB of random access memory. Every model converges well within one day of training.

3.5 Experimental Methodology

3.5.1 Description

Each experiment consists of a pre-training task meant to isolate a structural hypothesis about the target semantic parsing task. The pre-training task is called the *upstream*, and the target task is called the *downstream*. An experiment then runs in three phases.

In the first phase, a model learns structural representations of patterns in the upstream task through end-to-end training on an upstream dataset. In the second phase, the model weights are frozen,[4] and the embeddings are allowed to learn an alignment to the frozen structural representations inherited from the first phase. In the third phase, the model is evaluated for downstream performance against a test set. Formally, the experiment works as follows:

1. A transformer model Y of a fixed depth and hidden size is chosen and initialized. An upstream task U and a downstream task D are selected and given as a set of sentence pairings,[5] $U \subseteq S_U^* \times T_U^*$, $D \subseteq S_D^* \times T_D^*$. The downstream task is split into a training set D_{train}, and a test set D_{test}.
2. An embedding layer $e_U = (e_U^s, e_U^t)$ is prepended to the transformer model Y, and the model is trained with cross-entropy loss to convergence. We call this the *pre-training phase*.
3. The upstream embeddings e_U are removed and a new embedding layer, $e_D = (e_D^s, e_D^t)$, is initialized and prepended to Y. The weights of Y are frozen, but e_D is trained on D_{train} in order to fit a lexical alignment to the representations learned during pre-training. This phase is called the *fine-tuning phase*.
4. Finally, the composite model $Y \circ e_D$ (e_D having been fine-tuned) is paired with a (fixed) decoding algorithm and evaluated on D_{test} with respect to perplexity, accuracy, and NLC2CMD. This is called the *evaluation phase*.

A graphical description of the method can be found in Figure 3.1.

There are a few critical components of the proposed methodology that warrant discussion.

First, it is important that the model is pre-trained thoroughly[6] so that we may conclude that measured downstream performance is attributable to structural representations inferred from the upstream. It is our aim to classify the downstream task in terms of structural content, so we must ensure that downstream performance arises from structural content

[4]For weights to be "frozen" means they are not optimized during training.
[5]Also called "translation rules."
[6]Meaning trained all the way to convergence.

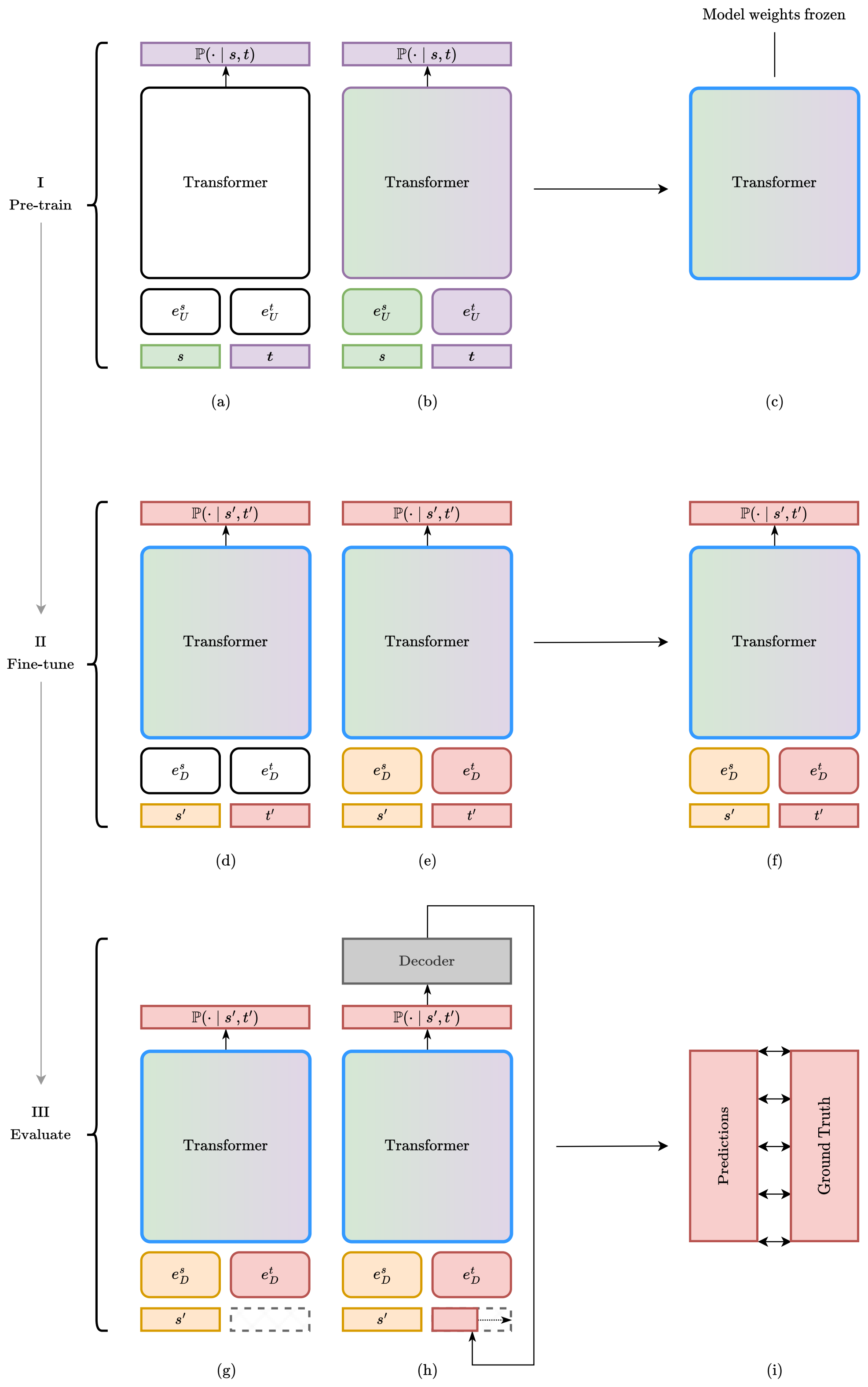


Figure 3.1: Experimental setup. During the pre-training phase (I), the model is randomly initialized (point (a)), then trained end-to-end on an upstream dataset (point (b)). The model weights are then frozen (indicated by blue border at point (c)). During the fine-tuning phase (II), a new embedding layer is initialized (point (d)) and trained while the transformer's weights remain frozen (point (e)). Finally, during evaluation (III), a decoding algorithm is used to generate predictions (point (h)) which are evaluated for perplexity, accuracy, and NLC2CMD against a ground truth (point (i)).

encoded in the transformer weights rather than spurious patterns which arise in incompletely trained models.

Second, the fine-tuning phase aligns tokens from the downstream vocabulary to the learned representations. As previously discussed,[7] tuning embeddings amounts to fitting a linear projection that best exploits the structure already learned into the model's weights. It cannot alter the model with higher order features learned from the downstream task. Since the vocabulary of the downstream is different from the vocabulary of the upstream, this alignment prevents meaningless predictions unrelated to the pre-trained structure of the model.

Third, the transformer weights are frozen during fine-tuning because we aim to measure structural overlap between the downstream and the upstream. Tuning the transformer weights during fine-tuning would allow leakage from the downstream task that would dilute the structural content imbued from the upstream. Naturally, this would make it difficult to determine whether observed downstream performance comes from the inductive bias of the upstream task or the representations learned from the downstream task (or a harmony of both). Therefore we freeze the transformer weights to isolate the learned inductive bias from downstream features.

We control for random fluctuations in training behavior by manually seeding any stochastic operations, such as model initialization. The project identified some discrepencies in behavior between PyTorch running in determinstic mode or non-deterministic mode. In order for results to be reliably reproducible, we opt to run all experiments in deterministic mode. More discussion on this can be found in Appendix A.1.

Furthermore, we fine-tune all models using the same learning rate and optimization parameters to control for potential conflations with learning rate.

3.5.2 Evaluation

We measure three performance heuristics in order to obtain a more complete view of transferability.

1. Perplexity gives a rough, language agnostic understanding of a model's ability to represent the target distribution. This is a useful heuristic but it leaves much to be known about the model's ability to produce the desired semantics.
2. To capture semantics more precisely, we also measure the NLC2CMD heuristic. NLC2CMD approximates accuracy normalized for flag order and attempts to capture semantic similarity in a more fine-grained way based on the syntactic structure of the Bash programming language.
3. Raw token accuracy is a third heuristic that captures lexical similarity and rewards some edge cases missed by NLC2CMD.

When considered together, these metrics give a holistic overview of the model's performance.

Comments on NLC2CMD

Though useful as a heuristic, this metric could be improved to capture some common structural equivalences in the bash programming language. This project found several areas of potential improvement beyond those mentioned by Agarwal et al. (2021).

[7]See Embeddings.

One such example is the relative similarity between the subshell and pipe operations. For example, the expressions "`basename $(pwd)`" and "`pwd | xargs basename`" are equivalent, yet the NLC2CMD metric would evaluate their similarity as -1.0, which is the worst possible score.

Another area of weakness is where commmands are conditionally equivalent depending on values passed as arguments or flags. For example, the `set` command lets the programmer configure the value of shell attributes. When used with the flag `-e` as in "`set -e`" (which appears in the test set), the result is the same as the longer invocation, "`set -o errexit`."[8] Therefore if the target is `set -e` and the prediction is `set -o errexit`, NLC2CMD would evaluate this incorrectly as -0.5 since the flag set is different and there is an additional argument in the prediction.

The NLC2CMD metric also fails to capture some equivalence of spacing. For example, the operator "`<( )`" takes the output of a list and groups it into a pipe that can be used as an argument in another command. In this case, the very simple command "`diff <(ls d1) <(ls d2)`" is equivalent to "`diff <( ls d1 ) <( ls d2 )`" semantically, yet the NLC2CMD metric would score these as -1.0. Similarly for the `$( )` operator.

In our case, Bash programs are tokenized by a bash *lexer* (which parses strings based on the Bash grammar) and encoded as space separated lists of tokens. This means a statement like "`<(ls dir)`" would parse as the token sequence "`<(`", "`ls`", "`dir`", "`)`". Furthermore, a model prediction comes in the form of a token sequence.

Token sequences are detokenized as a space separated list of tokens, so NLC2CMD's failure to admit equivalence in this case is attributable to the detokenizer rather than the model. Since the object of this thesis's study is the structural properties of the model, we did not spend time to fix the detokenizer to accommodate NLC2CMD and instead cross-reference other performance heuristics. We leave improvements to the NLC2CMD evaluation heuristic to future work, and hope this small study helps in the effort to improve the robustness of automated evaluation of Bash scripts.

3.6 English to Bash Dataset

Programming languages pose an interesting challenge to semantic parsing, and semantic parsers with programming language targets may assist developers in producing effective code.

Among programming languages, popular targets include SQL, Python, Java, Bash, and C#. SQL is an attractive target and has a rich literature. A large volume of data have been curated for tasks related to text to SQL, including SParC (Yu et al., 2019b), Spider (Yu et al., 2018), WikiSQL (Zhong et al., 2017), and CoSQL (Yu et al., 2019a). As discussed in Chapter 2, SQL has the additional complication of requiring a context besides the natural language input. This additional content comes in the form of a database *schema* which determines the set of valid SQL queries. This additional challenge is besides the interest of this thesis, so we have avoided this task.

Bash, however does not require additional context besides an input utterance in natural language. The output of Bash statements instead may depend on a latent context. For example, which files are available on the file system may change a program's exit value. We

[8]https://linuxcommand.org/lc3_man_pages/seth.html

alleviate this issue by masking context specific tokens such as filenames. This divorces the structural features of Bash from the latent execution context.

Technical Description

This dataset is adapted from the NLC2CMD competition hosted by NeurIPS 2020 (Agarwal et al., 2021), which itself was adapted from Lin et al. (2018) who originally introduced the dataset as "NL2Bash." The data consist of roughly 10,000 Bash one line commands paired with natural language annotations provided by professional Bash programmers.

An English and Bash tokenizer were provided with the NLC2CMD competition, and minor improvements were made by Fu et al. (2021), who produced the highest performing submission. To be consistent with the literature, these tokenizers were used in this thesis.

The English tokenizer uses the popular NLTK toolkit (Bird and Loper, 2004) to perform standard lexical normalizations common in natural language processing. These filter out common "stop" words with little semantic importance. They also normalize verb and noun endings: for example, different conjugations of the same verb are mapped to a canonical token associated with the verb. This is called *stemming*, and reduces the complexity of the token vocabulary with only a small semantic cost.

The Bash tokenizer is derived from the Bashlex Python package,[9] which is a version of the GNU Bash lexer transliterated into Python from C. Some normalizations are also made here, which include filtering regular expressions and file names to disaggregate the semantic structure of Bash programs from particular execution environments. These normalizations are distributed by Lin et al. (2018), and their implementation was used with only minimal adjustment.

[9] https://pypi.org/project/bashlex/

Chapter 4

Experiments

4.1 Baselines

4.1.1 Random Initialization

We would like to understand a baseline performance of the transformer model architecture before proceeding to measure any gains had by pre-training on linguistic structures. A transformer with randomized weights imbues little inductive bias beyond what's imposed intrinsically by the architecture itself. The embeddings learned must fit to the randomized landscape which is very high dimensional. In the context of langauge modeling, random sentence embeddings in a high dimensional space can match the performance of extensively pre-trained embeddings (Wieting and Kiela, 2019). This reinforces the importance of this baseline.

This is expected to be a lower bound on performance, since the weights have not adapted to any upstream structural patterns for exploitation by the embeddings.

Design

In this experiment there is no upstream dataset. The transformer model weights are initialized randomly and then frozen. For the reported baseline, we use uniform initialization. To select the hyperparameter α, we train a model for α set to every order of magnitude between 10^{-3} and 1. Additionally we train a model for every integer multiple of the order of magnitude with the best training performance. In our case, these values were 0.1 to 0.9. We then report the best performance of these models. This is called a *one-dimensional grid search*. The result is an approximation of the optimal hyperparameter value precise to one significant digit.

An alternative to the uniform initialization scheme is the Xavier scheme (Glorot and Bengio, 2010). For completeness, the experiment was repeated under the Xavier scheme. Since Xavier initialization is non-parametric, no hyperparameter tuning is necessary. The difference in performance between these schemes is surprising and discussed in Sections 5.1 and 5.3. In order to be consistent with the other experiments, the Xavier scheme for model initialization was chosen.

Since the results are inherently random, we run the experiment several times using different random seeds. No significant difference was observed between runs, which indicates stability in the methodology.

4.1.2 Copy Task

In order to ascertain the relevance of syntactic structures at the sentence level, it is useful to include a baseline which does not have syntactic structure. This means that the translation rule is reducible to a function of *words*, i.e. a *lexical map*, and the co-occurrence of words does not affect the lexical association.

Formally, where the translation rule f is a function, $f(s_1 \cdots s_n)_i = g(s_i)$ for all i for some function $g : S \to T$. We would like to avoid disturbing distributional parameters such as sentence length, token frequency, and word co-occurrence in order to isolate the property of lexical mapping as a baseline. Therefore we seek a function g which preserves these distributional properties of the source language. The simplest such g is the identity function, though a more interesting class of functions to consider is the set of *permutations*. These correspond to arbitrary bijective associations between words in a vocabulary.

In choosing a permutation for g, we arrive at a *copy task*, whereby tokens from the source sentence are "copied" to the target sentence, possibly passing through a deterministic word shuffling function on the way. Hence we have a baseline which capturing the effectiveness of a learned inductive bias for raw lexical alignment between source and target vocabularies. This motivates the following design for an upstream dataset.

Design

We avoid introducing an arbitrary token distribution since this may conflate with our results. Therefore, we synthesize a language which inherits the following distributional features from the downstream training set:

1. Word frequency
2. Sentence length
3. Word co-occurrence

To do this, we reassign the token values from the source language of the downstream training set randomly. The target sentences are then constructed similarly by picking a random permutation $\pi : S \to S$ and assigning words to their image under π. To be precise, the translation rule is $\{(s, \overline{\pi}(s)) : s \in S^*\}$ where $\overline{\pi}(s_1 \cdots s_n) := (\pi(s_1) \cdots \pi(s_n))$.

4.1.3 End-to-End Training

It is useful to estimate an upper-bound on performance attainable from this scheme. The downstream training set was constructed to share the distribution of the downstream set set, so we expect that a model trained on the downstream training set will have the best performance.

Therefore we train a model on end-to-end on the downstream training set, which is equivalent to pre-training on the downstream task (in the proposed methodology).
Including this baseline also serves as a useful check on our choice of model parameters. High accuracy and low perplexity during training indicates that our model architecture is expressive enough to model the training data.

Design

For this experiment we simply train the model to convergence using the Adam optimizer.

4.2 Within Task

Below are a few upstream tasks which are "within task" insofar as they have a formal language as their source or target.

4.2.1 English to Python

Syntactically different programming languages are often derived from a kernel of semantic patterns such as loops, control flow, hash maps, etc., which appear in many languages. These patterns are often called *idioms*. Pre-training on an upstream dataset which demonstrates these idioms may yield structural representations relevant to other semantic parsing tasks.

Python[1] is a general purpose scripting language used for filesystem manipulations similar to those common in Bash. However, the language is multi-paradigm and is frequently used in broader settings within software engineering and scientific computing. Beyond this, Python has a rich standard library and a mature ecosystem of packages supporting graphics, networking, simulation, and other computational tasks. An English to Python upstream task is therefore interesting because Python roughly covers a superset of semantic patterns frequently found in Bash.

Design

The English to Python dataset is derived from the CoNaLa bitext corpus (Yin et al., 2018). CoNaLa contains 2777 short Python programs mined from the StackOverflow website[2] paired with manually annotated statements of "intent" in natural language. These programs cover a broad variety of Python use-cases, including filesystem manipulation, control flow, error handling, and interaction with the NumPy (Harris et al., 2020) numerical computing library. The data are split into a training set with 2300 pairs and a test set with 477 pairs.

Tokenizer

Python, unlike Bash, has semantically significant whitespace. In particular, line terminators and indentation can define a logical block of code, introduce scope, break program syntax, and change the meaning of a program.

To be compatible with the OpenNMT toolkit, utterances must be encoded as space separated lists of tokens terminated by newline characters. Furthermore, in Python, string literal tokens may contain spaces within a syntactic unit which is incompatible with OpenNMT. The thesis found that most string literals in the CoNaLa dataset contained dummy content for the sake of illustration, which is common in sample code provided in answers to questions on the StackOverflow website. Because of this, the value of string literals have little semantic meaning in samples from CoNaLa, though they are syntactically important for producing grammatically correct Python code. Because of this, four tokens were introduced

[1]https://www.python.org/
[2]https://stackoverflow.com/

to the vocabulary: <NEWLINE>, <INDENT>, <DEDENT>, and <STRING_LITERAL>. These represent line terminator characters, indentation, dedentation (exit from a code block), and string literals respectively. For use in this project's experiments, a simple tokenizer was constructed to encode Python programs as token sequences compatible with OpenNMT. This system is invariant whether spaces or tabs are used for indentation. Table 4.1 demonstrates an example tokenization.

Program	`for i in range(256):`
	`    for j in range(256):`
	`        ip = ("192.76.%d.%d" % (i, j))`
	`        print(ip)`

Tokenization	`for i in range ( 256 ) : <NEWLINE> <INDENT> for j in range ( 256 ) : <NEWLINE> <INDENT> ip = ( <STRING_LITERAL> % ( i , j ) ) <NEWLINE> print ( ip ) <NEWLINE> <DEDENT> <DEDENT>`

Table 4.1: Sample Python tokenization.

4.2.2 English to SQL

SQL[3] is a domain-adaptable query language used to extract data from relational databases. The most common directive is the SELECT command, which specifies criteria against which a database backend filters and extracts data method agnostically—that is, without requiring the programmer to specify an index identifying target records. SQL may also be used to configure databases, update database topology, update records, and remove records; however, we restrict our study to utterances of the SELECT directive.

The most common utility in the downstream dataset is find, which makes up roughly two-thirds of the downstream dataset. The find command is structurally similar to SELECT insofar as it queries the filesystem, potentially applies filters (specified with flags), and returns the corresponding result. For this reason, we expect structural representations associated with constructing SELECT commands to transfer well to English to Bash.

Furthermore, SQL SELECT statements support hierarchical nesting which is directly analogous to the subshell (`$( )`) structure in Bash. Specifically, a (child) SELECT query may be the argument to the FROM keyword of an enclosing query, which executes the parent query on the result of the child query.

Finally, the dataset used (Yu et al., 2018) includes redundant target samples with different English labels. Formally, denoting the target language as SQL, the translation rule $f : S^* \to \texttt{SQL}^*$ is not injective. This demonstrates redundancy in natural language which may be modeled in weights.

Design

We use the Spider dataset (Yu et al., 2018), which contains roughly 10k manually annotated, complex and cross-domain SQL query, English sentence pairs. The data were provided by the HuggingFace[4] machine learning web platform, and we used the default train, test split.

[3]Short for "Structured Query Language."

[4]https://huggingface.co/datasets/spider

In order to prevent overfitting to database values present in training data, we mask particular values and numbers with a standard placeholder. This reduces the linguistic complexity of the task with minimal disturbance to its structure. A sample masking is shown in Table 4.2.

English	SQL
how many heads of department are over the age of 56	`select count(*) from head where age > 56`
how many heads of department are over the age of `_NUMBER`	`select count(*) from head where age > _VALUE`

Table 4.2: Query masking.

SQL has the additional challenge of modeling a latent context in the form of a database *schema*. The schema defines the relational topology of tables in the database, and this determines which queries are valid. For example, tables cannot be joined on non-existent keys, and non-existent columns cannot be queried.

We elect to avoid this problem by omitting any additional table context entirely. Fortunately, we find that when annotations and queries are masked, this poses no significant challenge to our model, which successfully learns the task and converges with over 82% accuracy on the test set.

4.2.3 SQL to English

We would like to understand whether the *direction* of translation imposes structural differences relevant to transfer. To this end, we study an example of a reversed semantic parsing task where the input is a statement in formal language and the target is a description in natural language.

Natural language is inherently collision-prone, which means several natural language sentences should be likely targets in the conditional distribution output from an appropriately trained translation model. We expect this to pose a challenge to our model, so we seek a dataset which illustrates this redundant property of natural language.

As mentioned above, the translation rule in the English to SQL task is non-injective. Therefore the induced translation rule from SQL to English is not a function. This precisely exposes the collision property of natural language we aim to extract. Additionally, the forward direction from English to SQL will give an informative benchmark against which we can evaluate the relevance of translation direction to the effectiveness of transfer.

Design

The task is constructed by flipping every sentence pair in the Spider dataset so that SQL queries are passed as inputs and the English annotations are used as targets.

4.3 Natural Languages

4.3.1 English to German

Programming languages are usually designed to reflect patterns in natural language. This facilitates their acquisition by humans who are familiar with natural language. Natural language can be sourced in large quantities from the internet, and there has been a substantial effort to curate large datasets of natural language sentence pairs for use in automated translation systems for decades.

Models trained on large corpi on unstructured natural language data (called *freetext*) have demonstrated strong multi-task performance (Raffel et al., 2020; Brown et al., 2020) which has even generalized to formal contexts wherein transformers pre-trained on natural language data are used to perform general-purpose computations (Lu et al., 2022).

The performance of these models in formal contexts is partially attributable to code data present in their training distributions, so it is interesting to study the generalizability of structural representations specific to natural language data using a datasets that is known to obtain no code.

Furthermore, there was a large quantity of English to German bitext introduced at the WMT14 workshop.[5] Using this gives us a chance to study a dataset which is considerably larger than the others in this project without exceeding the expressiveness of our simple model.

Design

We use the English to German task published at WMT'14 which has now become a popular benchmark in the field of neural machine translation. In particular, we use the data curated by Luong et al. (2015), which includes over 4.4M sentence pairs aggregated from several sources extracted from news articles in English and German.

4.4 Formal Sequence Transduction

4.4.1 Infix $\leftrightarrow$ Postfix $\leftrightarrow$ Prefix

We would like to study the concept of redundancy more closely and examine a pattern where redundancy is required for grammatical correctness. It is also preferable to choose a task which does not disturb the distributional features of the source and target, and one which does not have a substantial difference in vocabulary and distribution across source and target. Lateral translation between expressions in infix, prefix, and postfix notation satisfy these properties.

Here, an expression is in *prefix* notation if an operand precedes its arguments as in `(+ 1 2 3)`, it is in *postfix* notation if an operands follows its arguments as in `(1 2 3 +)`, and it is in *infix* notation is the operand is placed between arguments as in `(1 + 2 + 3)`. Translations between these representations preserve distributional parameters of the argument tokens, and must inject redundancy when interfacing with infix notation. In this case, the model must learn that although the distribution of operands differs between source and target, the semantics remain fixed.

[5] https://www.statmt.org/wmt14/

Design

Three languages were synthesized from a vocabulary V of 26 terms and a vocabulary O of 4 operands. The sentences are constructed using the following algorithm:

1. Select a number of terms d from a Poisson distribution whose mean matches the mean sentence length of of the downstream training set.
2. Sample d terms uniformly at random from V and sample an operation o uniformly at random from O.
3. Prefix, postfix, and infix o among the terms sampled from V.

For example, if $d = 4$, we may first sample `c`, `p`, `b`, `t` from V, `+` from O, and produce the sentences "`+ c p b t`", "`c p b t +`", "`c + p + b + t`". 6 machine translation tasks were then constructed from the 3 datasets by matching corresponding utterances from every pairwise choice of distinct languages.

4.4.2 Reversal Task

In a reversal task, the source and target languages share distributional parameters: the token co-occurrence statistics are identical between source and target and the the sentence length distribution is the same. This task is interesting because the translation rule cannot be inferred lexically and depends on the entire input sentence. It is not stable under permutation of the inputs, so this imbues a bias whereby the model must attend to the *order* of tokens, rather than their lexical content or co-occurrence statistics alone.

Interestingly, however, this bias is *anti*-sequential insofar as the first half of the target sentence is independent from the first half of the input sentence. This imbues a bias in the model to attend from right to left rather than from left to right. Therefore the resulting performance is indicative of a left-to-right structure in the downstream task.

Design

The consist of 8277 training pairs and 2030 test pairs of procedurally generated token sequences. As in the infix, prefix, postfix task, the sentence length was sampled from a Poisson distribution with the same mean as the average sentence length in the downstream training set. The tokens were then sampled randomly from a corpus of random strings to fill the sentence. To control for token distribution, one baseline was used where tokens were sampled uniformly. Another baselines was used where tokens were sampled from the marginal distribution of tokens inherited the source language of the downstream task, which is roughly Zipf-like. Given a source sentence, a target sentence was constructed by reversing the order of tokens in the source sentence.

Chapter 5

Results

5.1 Training Curves

The training curves for the fine-tuning phase for several upstreams are shown in Figure 5.1. The curves are clearly framed above by the end-to-end training, and below by the uniformly initialized model (as expected). Beyond this, we find three clusters in the training accuracy:

- The highest accuracy cluster is populated by the within-task upstreams, which include the SQL tasks and English to Python.
- The next highest band is occupied by the copy task, the reversal task, and the Xavier initialized model, with German lagging slightly behind.
- Finally, the prefix to infix task is comparatively poor.

Thus, we find that the inductive biases learned within task are easily exploitable and model the training set well.

Test Accuracy

Most models quickly improve accuracy on the test set. The SQL tasks converge the quickest, and post strong performance on both the training and test sets. This indicates an effective transfer of generalizable structural representations between tasks. The English to German dataset trails behind the others but eventually obtains the highest test accuracy of all upstreams.

Thus, the English to German bias induces poorer performance on the training set but better generalizing performance on the test set. This behavior usually indicates that injected bias effectively regularizes the task and is preventing overfitting to the training set. Seeing such behavior in this case is surprising because only the embedding layers, which are *linear*, are learned during this phase of training. Nonetheless, overfitting can still happen in linear models when the feature space is large and there more features than training samples. In this case the feature space is possibly very large indeed, and there are far more parameters than there are downstream training samples. These findings suggest that learned inductive biases may be used in conjunction with freezing to regularize models.

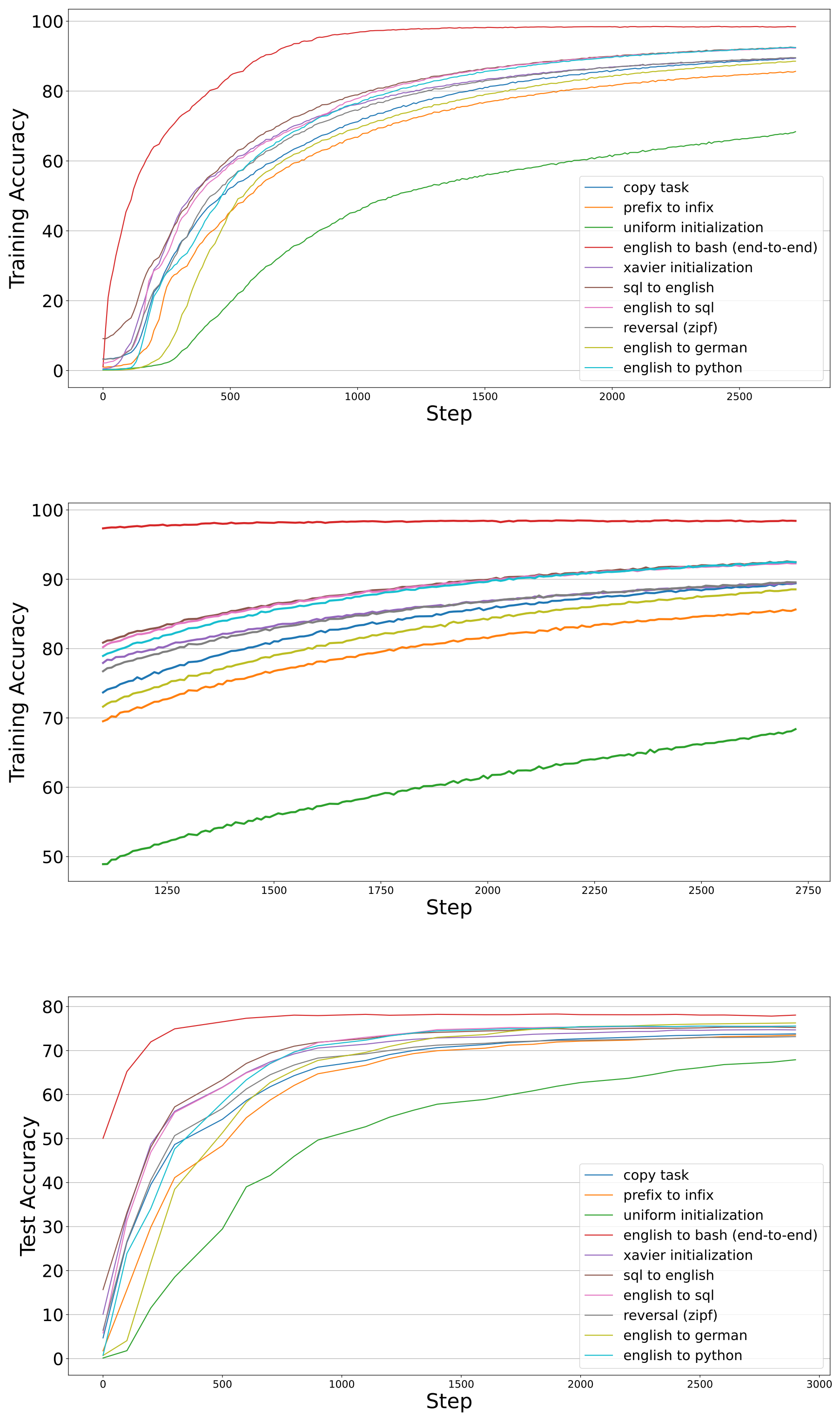


Figure 5.1: Training curves. (top) training accuracy during the fine-tuning phase for several upstreams; (middle) the last several iterations from the training accuracy curves enlarged; (bottom) test accuracy computed at several checkpoints.

Random Models

The random model initialized under the uniform scheme performs performs poorly as expected. Not only is the inflection of the training accuracy curved delayed, but it only slowly improves and to a modest cap. This can be explained by the absence of structural bias present in the model.

Rather surprisingly, the Xavier initialized model performs impressively well. It appears in the second cluster of training accuracies and outperforms almost half of the models when evaluated for test accuracy. This indicates that some structural biases may actually cause *negative* transfer when compared to a randomly initialized model.

Negative transfer could be explained by sparsity induced in the transformer weights by training on the upstream. To elaborate, training the transformer weights exposes them to updates, which may cause some weights to grow large in magnitude with respect to the others. Once the weights are frozen, this has the effect of reducing the variance and increasing the bias of the model thereby constraining the functions learnable by the embeddings during fine-tuning. If the pre-trained biases don't transfer well to the target task, it may be more effective to allow embeddings to fit to spurious correlations in a randomly initialized model.

Nonetheless, it's suprising that the Xavier model outperforms some of the sequentially biased upstreams (including the prefix to infix and English to German) on training accuracy since the downstream task appears to be sequential and the transformer model does not have intrinsic sequential bias.

Interestingly, all models converge to a similar (and respectable) test accuracy. Table 5.1 gives a fine-grained summary of these results.

5.2 Metrics

	Bash		
Experiment	ppl	Accuracy	NLC2CMD
Uniform Init.	3.29	67.9	−8.3
End-to-End	3.80	78.1	16.5
En-Python	3.39	74.2	10.1
En-SQL	3.14	75.6	**14.1**
SQL-En	3.27	75.2	13.5
En-De	**2.77**	**76.3**	12.9
Prefix to Infix	3.32	73.5	8.7
Reversal	3.33	73.2	7.2
Copy	3.24	73.8	11.1

Table 5.1: Results. (*NLC2CMD scaled by 100*)

The NLC2CMD metric is our touchstone for semantic equivalence. Interestingly, while the English to German upstream had the best performance with respect to perplexity and accuracy, it did not outperform every task on NLC2CMD. Indeed, there seems to be little correlation among desirable perplexity scores and desirable NLC2CMD evaluations. A scatter plot is displayed in Figure 5.2.

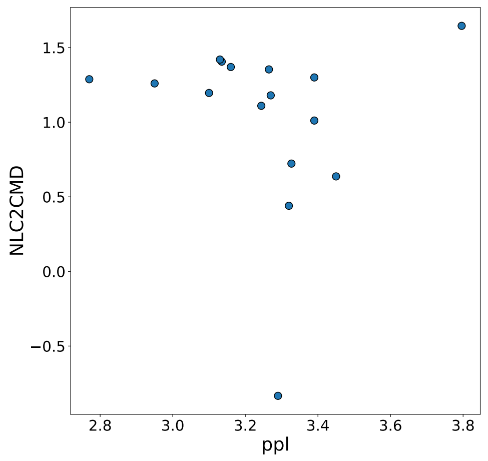


Figure 5.2: Perplexity vs NLC2CMD. The perplexity and NLC2CMD for 16 fine-tuned models measured on the English to Bash task. Measured correlation is -0.089.

A small negative correlation is observed, though there is little discernable pattern between the two metrics. This indicates that perplexity is a poor heuristic for semantic accuracy in the context of Bash programs, and may be a misleading heuristic in the context of semantic parsing in general. These findings are similar to results of Tran et al. (2019), which demonstrate that BLEU score, though a popular score in the machine translation literature, is an invalid metric of semantic similarity in the context of computer code. To generally evaluate effectiveness requires a careful study of program traces, as is highlighted by Churchill et al. (2019). These findings corroborate this idea by demonstrating an example where distributional similarity is a poor predictor of semantic similarity.

Nonetheless, the low perplexity of the English to German task indicates a strong ability to model the distributional characteristics of the downstream, however, lower NLC2CMD compared to the SQL task indicates a failure to capture the fine-grained structure of Bash. This is evidence that there are structural patterns in the English to SQL task that are relevant to Bash and absent in the English to German task.

English to SQL is the highest scoring upstream, which suggests a strong structural similarity between the two tasks. Rather surprisingly, the inverted task mapping SQL queries to English annotations performed similarly. This suggests that transformers learn semantic representations which embed structural features as *unordered pairs* of sentences, and are somewhat robust to directionality of translation tasks. Further experimental scrutiny of this claim would be an interesting direction of research.

Also surprising is the rather poor performance of English to Python as an upstream task. Its performance is matched (and even outpaced) by the copy task. This coaxes out an inherent lexical structure in the downstream Bash task. In particular, it shows that *lexical mapping alone makes up a substantial portion of the difficulty of the task*. Perplexity, accuracy, and NLC2CMD performance are all comparable between the copy and Python semantic parsing tasks. This indicates that although English to Python is similar to the downstream on the *task* level (insofar as both tasks are semantic parsing tasks), in reality there is little marginal gain using structural representations suitable for English to Python over representations suitable for learning simple lexical alignments.

Another item of interest is the reversal task. We hypothesize that the downstream task is sequentially biased, so we expect any of the sequential tasks to see a positive performance gain. The reversal task was designed with an *anti*-sequential structure, whereby the first half of the output is independent from the first half of the input. Therefore the observably poor performance of the reversal task supports the hypothesis that the Bash semantic parsing task has an inherently sequential structure whereby tokens at the end of the input sequence have little relevance to which tokens are included at the beginning of the target sequence.

The prefix, postfix, infix task contributed little to downstream performance. The experiment was repeated for every pair of parallel data. None were significantly more performant than the others, so prefix to infix is reported in Table 5.1 as the best.

5.3 Random Baselines

Looking back to Figure 5.1, we see that the model with Xavier initialized weights performs surprisingly well. This can be explained by *spurious correlations* in the weights which can be exploited by embeddings. That is, randomly initializing a model with appropriately designed noise is bound to produce emergent patterns by chance.

These chance patterns can be adapted to by the embeddings, and for a large enough space these patterns may appear somewhat reliably, even with several different seeds. To illustrate this, four Xavier initialized models were fine-tuned, each with different dropout values.

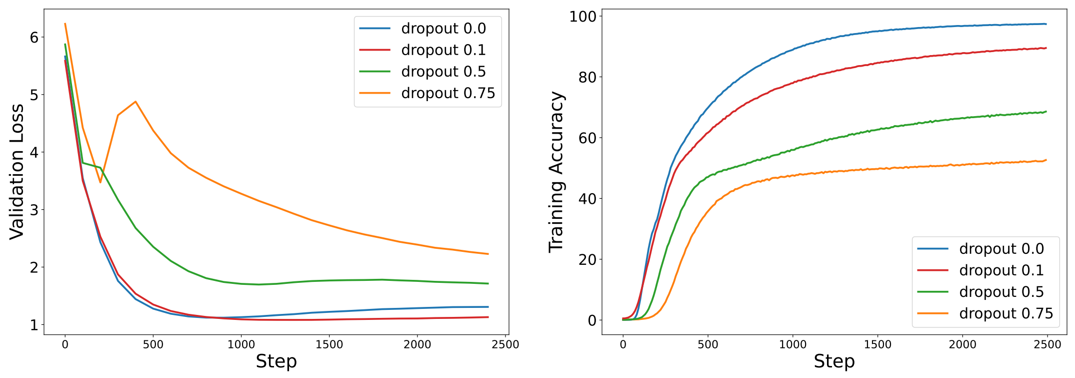
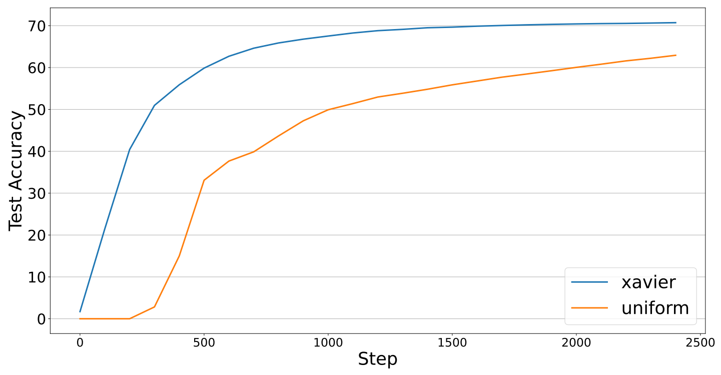


Figure 5.3: Spurious performance in random models. Training curves from fine-tuning random models with frozen, Xavier initialized weights for four different dropout values.

Dropout randomly masks input values to zero throughout the neural network. A higher dropout rate increases the probability of a given input being masked, which decreases the likelihood of coordinated weight patterns (Srivastava et al., 2014). Accordingly, adding dropout throughout the transformer model destabilizes the visibility of weights from the perspective of the embeddings, and makes it more difficult to fit to spurious patterns. Figure 5.3 clearly shows the increasing difficulty with higher dropout rates, which highlights the reliance on spurious correlations for performance. The immediate jump in validation loss early in the training of the model with a dropout rate of 0.75 likely corresponds to an embedding realignment as a result of a spurious pattern being masked.

Curiously, the validation loss of the model with no dropout *increases* after a trough around step 800 though training accuracy and training loss (not displayed) continue to improve. This is evidence of *overfitting*, which is surprising since the transformer weights are frozen, which reduces the complexity of the model massively. This suggests that randomly projecting sequence embeddings into a high-dimensional space may substantially reduce the complexity of this task. A similar phenomenon was observed by Wieting and Kiela (2019) in the context of language modeling, which reinforces the importance of considering randomized baselines and draws attention to the importance of appropriately selecting initialization schemes.



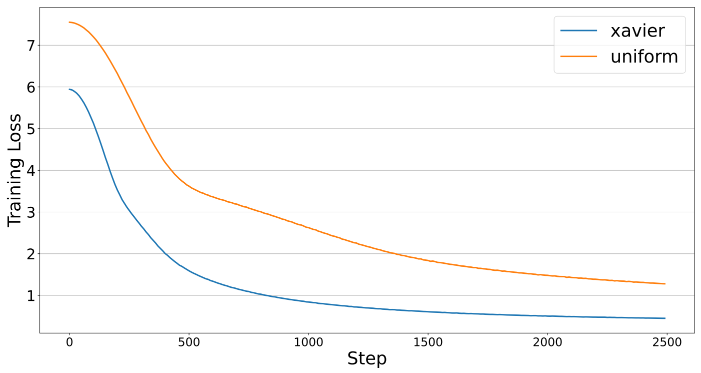


Figure 5.4: Random initializations. Performance evaluated at several checkpoints for a Xavier initialized model (blue) and a uniformly initialized model (orange). (top) Test accuracy; (bottom) training loss.

Along these lines, we reconsider the comparison between the Xavier and uniformly initialized models. We plot training curves for the fine-tuning of a uniformly and a Xavier initialized model in Figure 5.4.[1] The dropout rate is 0.1.

The Xavier scheme clearly outperforms the uniform scheme. The difference can be explained by improved normalization of weight magnitudes between components of the transformer under the Xavier scheme. The hidden dimension in our case is 512, the feedforward dimension is 2048, and the key and value dimensions are 64. The projection tensors therefore vary in dimension by 3 orders of magnitude. This may cause vast differences in activation magnitudes if all weights are initialized in the same range. The high performance of the Xavier initialized model shows that the gradient stabilizing effect of Xavier initialization demonstrated by (Glorot and Bengio, 2010) also improves information flow through transformer-based semantic parsers and may induce spurious patterns that effectively represent potentially complex tasks.

5.4 What about compute?

German		Bash		
Samples	Steps	ppl	Acc.	NLC2CMD
10k	2500	3.39	75.0	13.0
10k	7500	3.27	75.0	11.8
100k	2500	3.13	76.0	**14.2**
100k	7500	3.16	75.8	13.7
4.4M	2500	2.95	76.0	12.6
4.4M	10000	2.77	76.3	12.9

Table 5.2: English to German, several compute samples.

Now we returning to the other upstream tasks. The delayed inflection in the training curve and high overall downstream test accuracy obtained by the English to German dataset indicate regularization caused by a bias introduced during pre-training. This could be due to te structure of the German task, or it could be due to the additional compute expended during pre-training.

The English to German dataset is considerably larger than the other upstreams. It has 4.4M training pairs, with an average sentence length of 26 words whereas the other datasets have roughly 5k to 10k training pairs with average sentence length around 12 words. It takes over 3 times longer to train to (upstream) convergence on English to German than on the other tasks. This leads to the question: *is the performance gain due to the structural features of the English-German MT task, or to the additional compute expended during pre-training?*

Compute can be measured in terms of dataset size, number of training iterations, model size, and parallelism. The last two features are controlled closely in our methodology, but the former two are not. To test the dependence on training iterations, we run a further experiment with the dataset randomly downsampled to 10k samples and 100k samples, and we run models for a variable number of training steps. We then reevaluate performance. Results are shows in Table 5.2

[1]Shown are the curves associated with the random seed of 9001. The experiment was repeated for several seeds and no significant difference was observed between runs.

The perplexity and accuracy clearly trend towards better performance with more compute. Surprisingly, the NLC2CMD metric does not share the same trend. The model trained on 100k samples from the upstream for 2500 training steps was stopped before convergence and now matches the performance of the highest performance upstream, English to SQL.

While interesting, this does not degrade the conclusion from Table 5.1 that English to Bash is structurally closer to English to SQL than English to German. This is because the smaller German upstream which matches performance with the English to SQL model was stopped before convergence, so the weights surface does not accurately represent the structure of the English to German task.

However, *this does suggest that more training does not always lead to better semantic representations on an out-of-distribution downstream task*, and that it may be effective to regularize the bias imbued by any single structural pattern in upstream pre-training data. These findings suggest that there is a certain "sweet spot" after which additional training may overfit to upstream structures.

Chapter 6

Discussion

6.1 Critical Evaluation of Method

Perhaps the most substantial contribution of this thesis beyond the methodology proposed by Papadimitriou and Jurafsky (2020) is adaptation to the sequence transduction domain. Furthermore, evaluating performance against metrics besides perplexity reveals the complexity inherent to assessing semantic equivalence of computer programs.

This methodology is an effective method for measuring structural overlap in terms of downstream modeling performance of structural representations learned during pre-training. It can be used to classify downstream tasks in terms of relevant upstream structural patterns that transfer to the target domain. This, in turn, may be used to improve pre-training strategies or understand the mechanics of transfer learning more deeply. But to classify the natural language to Bash task conclusively would require a deeper study.

Methodological Weaknesses

The most obvious weakness in the proposed methodology is a poor evaluation scheme. While NLC2CMD is a thoughtful heuristic for program similarity, it does not reliably determine semantic equivalence of Bash statements. A more robust system is needed to conclusively determine the semantic accuracy of predicted sentences.

Another problem of the proposed methodology is the requirement to specify structural patterns at the *task* level. That is, the results presented in this project do well approximating an answer to the proposed question as to which *particular* structural inductive biases transfer to the downstream task, though it cannot answer the question more precisely than determining similarity in task. This is not problematic for some of the artificial tasks proposed which were carefully constructed to illustrate abstract structural patterns. For the other tasks however, including the natural machine translation and semantic parsing upstream tasks, the structural patterns are merely *inferred* based on hypotheses of the researcher.

To obtain a more precise structural classification would require researchers to carefully construct artificial tasks isolating each pattern existing in the upstream tasks. Unfortunately, this is difficult and prone to errors. Instead, an automatic inferential method is a more practical. Some headway has been made in this area toward extracting implicit grammars from language freetext and bitext. This is often referred to as *grammar induction*

For example, Shin et al. (2019) propose a system for mining programming idioms from a corpus of computer code, which may be extended to include other upstream patterns. Such automated structural extraction systems are interesting because they are more amenable to end-to-end training systems than manual task definitions.

Another weakness of the method is limitation to semantic reconstruction. Beyond semantic performance, it is interesting to also understand training performance. In this thesis, for example, we found that the English to German upstream had effective structural representations to which downstream embeddings could adapt, however the model took longer to converge. Practical considerations of time to convergence on the downstream task would be relevant to applications of the proposed methodology.

For example, an area of scientific and practical interest is the construction of semantic parsers for low-resource sources or targets. In these contexts, it is highly relevant which downstream data samples are required for satisfactory performance since the downstream samples available may be limited or absent. In the context of the SQL target, for example, augmenting the training set with synthesized training examples has shown promising results (Yu et al., 2021a,b; Shi et al., 2021). These systems rely on manually defined pairings between patterns in English and patterns in SQL which define what's called a *synchronous context-free grammar* (SCFG). SCFGs are expensive to construct, and it would be useful to identify which rules require inclusion in the synthesis scheme, and how many samples should be generated. Results from this thesis indicate that it is not always better to include more pre-training samples.[1]

Finally, another interesting weakness in this methodology is the uncontrolled compute allocated to the upstream phase. Electing to use off-the-shelf upstream tasks unfortunately exposes the method to unintended variation in dataset size which may conflate with results. In this project, most upstream datasets were selected to be similar in size to control for computational discrepancy. With the exception of the English to German, all upstream tasks converged within a similar amount of time on the same hardware, and all models (including English to German) converged using the same model architecture and standard regularizations. In practice, we cannot expect upstream datasets to yield to uniform training strategies equipped with similar amounts of compute. The proposed methodology does not account for differences required for upstream convergence.

For example, the current setup does not control for differences in compute in terms data size. This could be considered in the measurement for the effectiveness of transfer. It may be the case that structures which require more compute to learn may not be worth the tradeoff. Which structures are easier to learn and which are more difficult? This question remains unanswered but would be interesting to include in a discussion on the effectiveness of transfer. Furthermore, more compute could result in more pronounced inductive biases, though we have no way to measure how pronounced an inductive bias is. Do statistics about the weights suffice, or do we need to study individual values?

Further Work

It would be interesting to unfreeze the transformer weights during fine-tuning and compare convergence rates between upstream tasks. In this setting, pre-training is used as a form of intelligent model initialization. Initializing the model with *appropriate* biases could allow larger weights at initialization time, which may lead to convergence over fewer training

[1]See section 5.4.

iterations and with less data in total. These models may be well suited to low-resource settings.

Seeing the possibility of negative transfer in this study, it would be interesting to see if early stopping on the upstream (before convergence) could improve downstream performance. This is a way to approximate weakening the inductive bias while still imbuing it slightly.

Of course, having a reliable measure of execution accuracy would also be useful.

6.2 Conclusions

- We set out to study the question: *which structural inductive biases measurably transfer to semantic parsing?*
- We proposed a method for determining an answer to this question and applied it to English to Bash, using a few example structural datasets as upstreams.
- We found that English to Bash is structurally similar to English to SQL, and is strongly similar to lexical mapping.
- More training on upstream datasets which are out of distribution from the downstream target may lead to stronger perplexity and accuracy, but may not produce better semantic representations. Therefore it may be valuable to regularize the distribution of structural patterns in the upstream dataset during pre-training.
- This thesis shows that although English to Bash is not as structurally similar to English to German, a partial inductive bias from English to German may effectively represent semantic structure of English to Bash better than a complete inductive bias, and may even match performance of an upstream which is more structurally similar to the downstream.
- The initialization scheme matters, and Xavier initialization alone has a rich enough weights surface for embeddings to find spurious patterns that represent English to Bash surprisingly well.

References

M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016. URL https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf.

M. Agarwal, T. Chakraborti, Q. Fu, D. Gros, X. V. Lin, J. Maene, K. Talamadupula, Z. Teng, and J. White. Neurips 2020 nlc2cmd competition: Translating natural language to bash commands. In H. J. Escalante and K. Hofmann, editors, *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*, volume 133 of *Proceedings of Machine Learning Research*, pages 302–324. PMLR, 06–12 Dec 2021. URL https://proceedings.mlr.press/v133/agarwal21b.html.

S. Bird and E. Loper. NLTK: The natural language toolkit. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, pages 214–217, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL https://aclanthology.org/P04-3031.

T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

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

B. Churchill, O. Padon, R. Sharma, and A. Aiken. Semantic program alignment for equivalence checking. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, page 10271040, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367127. doi: 10.1145/3314221.3314596. URL https://doi.org/10.1145/3314221.3314596.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *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)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

Q. Fu, Z. Teng, J. White, and D. C. Schmidt. A transformer-based approach for translating natural language to bash commands. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1245–1248, 2021. doi: 10.1109/ICMLA52953.2021.00202.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL https://proceedings.mlr.press/v9/glorot10a.html.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

E. Grefenstette, P. Blunsom, N. de Freitas, and K. M. Hermann. A deep architecture for semantic parsing. In *Proceedings of the ACL 2014 Workshop on Semantic Parsing*, pages 22–27, Baltimore, MD, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-2405. URL https://aclanthology.org/W14-2405.

C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer. Mapping language to code in programmatic context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1643–1652, Brussels, Belgium, Oct.-Nov. 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1192. URL https://aclanthology.org/D18-1192.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015. URL http://arxiv.org/abs/1412.6980.

G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. Rush. OpenNMT: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*, pages

67–72, Vancouver, Canada, July 2017. Association for Computational Linguistics. URL https://aclanthology.org/P17-4012.

M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.703. URL https://aclanthology.org/2020.acl-main.703.

X. V. Lin, C. Wang, L. Zettlemoyer, and M. D. Ernst. NL2Bash: A corpus and semantic parser for natural language interface to the linux operating system. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA). URL https://aclanthology.org/L18-1491.

K. Lu, A. Grover, P. Abbeel, and I. Mordatch. Frozen pretrained transformers as universal computation engines. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36 (7):7628–7636, Jun. 2022. doi: 10.1609/aaai.v36i7.20729. URL https://ojs.aaai.org/index.php/AAAI/article/view/20729.

T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, Sept. 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1166. URL https://aclanthology.org/D15-1166.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space, 2013. URL https://arxiv.org/abs/1301.3781.

M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

I. Papadimitriou and D. Jurafsky. Learning Music Helps You Read: Using transfer to study linguistic structure in language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6829–6839, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.554. URL https://aclanthology.org/2020.emnlp-main.554.

J. Pennington, R. Socher, and C. Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, Oct. 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1162. URL https://aclanthology.org/D14-1162.

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

P. Shi, P. Ng, Z. Wang, H. Zhu, A. H. Li, J. Wang, C. Nogueira dos Santos, and B. Xiang. Learning contextual representations for semantic parsing with generation-augmented pre-training. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(15):13806–13814, May 2021. URL https://ojs.aaai.org/index.php/AAAI/article/view/17627.

E. C. Shin, M. Allamanis, M. Brockschmidt, and A. Polozov. Program synthesis and semantic parsing with learned code idioms. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/cff34ad343b069ea6920464ad17d4bcf-Paper.pdf.

K. Sinha, R. Jia, D. Hupkes, J. Pineau, A. Williams, and D. Kiela. Masked language modeling and the distributional hypothesis: Order word matters pre-training for little. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2888–2913, Online and Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.230. URL https://aclanthology.org/2021.emnlp-main.230.

S. Smith, M. Patwary, B. Norick, P. LeGresley, S. Rajbhandari, J. Casper, Z. Liu, S. Prabhumoye, G. Zerveas, V. Korthikanti, E. Zhang, R. Child, R. Y. Aminabadi, J. Bernauer, X. Song, M. Shoeybi, Y. He, M. Houston, S. Tiwary, and B. Catanzaro. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model, 2022.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

W. Su, X. Zhu, Y. Cao, B. Li, L. Lu, F. Wei, and J. Dai. Vl-bert: Pre-training of generic visual-linguistic representations. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SygXPaEYvH.

C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016. doi: 10.1109/CVPR.2016.308.

N. Tran, H. Tran, S. Nguyen, H. Nguyen, and T. N. Nguyen. Does bleu score work for code migration? In *Proceedings of the 27th International Conference on Program Comprehension*, ICPC '19, page 165176. IEEE Press, 2019. doi: 10.1109/ICPC.2019.00034. URL https://doi.org/10.1109/ICPC.2019.00034.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

J. Wieting and D. Kiela. No training required: Exploring random encoders for sentence classification. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=BkgPajAcY7.

P. Yin, B. Deng, E. Chen, B. Vasilescu, and G. Neubig. Learning to mine aligned code and natural language pairs from stack overflow. In *International Conference on Mining Software Repositories*, MSR, pages 476–486. ACM, 2018. doi: https://doi.org/10.1145/3196398.3196408.

T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, 2018. Association for Computational Linguistics.

T. Yu, R. Zhang, H. Er, S. Li, E. Xue, B. Pang, X. V. Lin, Y. C. Tan, T. Shi, Z. Li, Y. Jiang, M. Yasunaga, S. Shim, T. Chen, A. Fabbri, Z. Li, L. Chen, Y. Zhang, S. Dixit, V. Zhang, C. Xiong, R. Socher, W. Lasecki, and D. Radev. CoSQL: A conversational text-to-SQL challenge towards cross-domain natural language interfaces to databases. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1962–1979, Hong Kong, China, Nov. 2019a. Association for Computational Linguistics. doi: 10.18653/v1/D19-1204. URL https://aclanthology.org/D19-1204.

T. Yu, R. Zhang, M. Yasunaga, Y. C. Tan, X. V. Lin, S. Li, I. L. Heyang Er, B. Pang, T. Chen, E. Ji, S. Dixit, D. Proctor, S. Shim, V. Z. Jonathan Kraft, C. Xiong, R. Socher, and D. Radev. Sparc: Cross-domain semantic parsing in context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy, 2019b. Association for Computational Linguistics.

T. Yu, C.-S. Wu, X. V. Lin, bailin wang, Y. C. Tan, X. Yang, D. Radev, richard socher, and C. Xiong. GraPPa: Grammar-augmented pre-training for table semantic parsing. In *International Conference on Learning Representations*, 2021a. URL https://openreview.net/forum?id=kyaIeYj4zZ.

T. Yu, R. Zhang, A. Polozov, C. Meek, and A. H. Awadallah. SCore: Pre-training for context representation in conversational semantic parsing. In *International Conference on Learning Representations*, 2021b. URL https://openreview.net/forum?id=oyZxhRI2RiE.

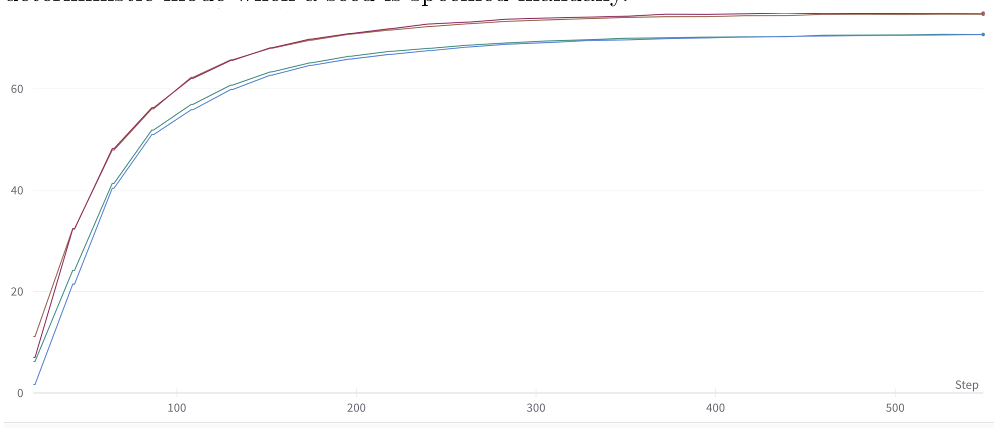
V. Zhong, C. Xiong, and R. Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103, 2017.

Appendix A

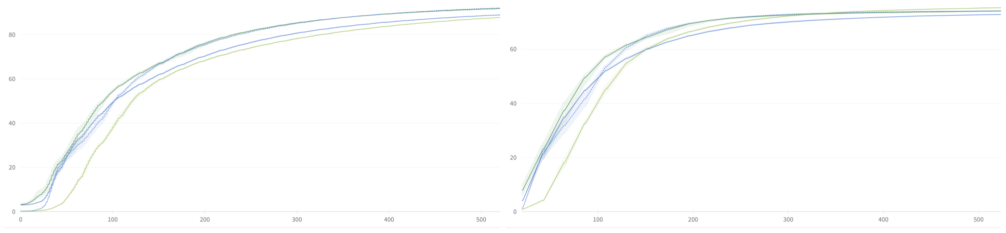
Other Work

A.1 Manually Specified Seeds

- Later in the thesis, when testing the robustness of results, it was discovered that there is a significant difference between running experiments with the CuDNN backend in *deterministic* mode or *non-deterministic* mode.
- The CuDNN backend dynamically selects algorithms implementing common operations in deep learning based on properties of the device and input data. Some implementations assume the cost of non-determinism to improve performance. In deterministic mode, CuDNN avoids such implementations and prefers deterministic algorithms.
- For reproducibility from a random seed, it is essential that CuDNN be operated in deterministic mode when a seed is specified manually.



- The fine-turning phase for German, Python, SQL, and Copy tasks were each repeated several times in non-deterministic mode to measure the robustness to this feature of CuDNN.



- The figure shows that the relative pattern of the training trajectories are roughly the same as in the deterministic case, though the values differ slightly.
- To investigate the details as to how and when these modes would be a distraction from the main idea of this thesis.
- Therefore, to promote reproducibility, this thesis has opted to report all results from runs with CuDNN set in deterministic mode, and all experiments are run with the random seed 9001 unless explicitly repeated with different seeds in tests of robustness.

A.2 Testing of Tellina Evaluation

- Before discovering the NLC2CMD metric, the project attempted to adapt the several heuristics proposed by Lin et al. (2018).
- Lin et al. (2018) are responsible for collecting the data which eventually matured into those data used in the NLC2CMD competition, and originally proposed a heuristic-based evaluation scheme called *template accuracy*.
- Several bugs in the distributed implementation were discovered, including an error which unnecessarily aggregated commands with the same utility, which caused a critical issue mismatching statements in the submitted list of predictions.
- These bugs were suggested to Lin et al. (2018) by the project.[1]
- Beyond this, an error was discovered in the published implementation of template accuracy calculations which failed to identify content in subqueries calling external scripts as syntactically correct Bash.
- This caused the suite to reject over 25% of the library's devlopment and test sets and over 20% of the training set during computations of template accuracy, which is a rough predecessor of NLC2CMD.
- Furthermore, an error was discovered (and fixed) by the project whereby BLEU scores were incorrect by up to 4% due to incorrect accounting for white space.
- In all, just over 1 week was spent attempting to rectify these evaluation issues before discovering the NLC2CMD system.

[1]https://github.com/TellinaTool/nl2bash/issues/33

Appendix B

Model

See Figure B.1.

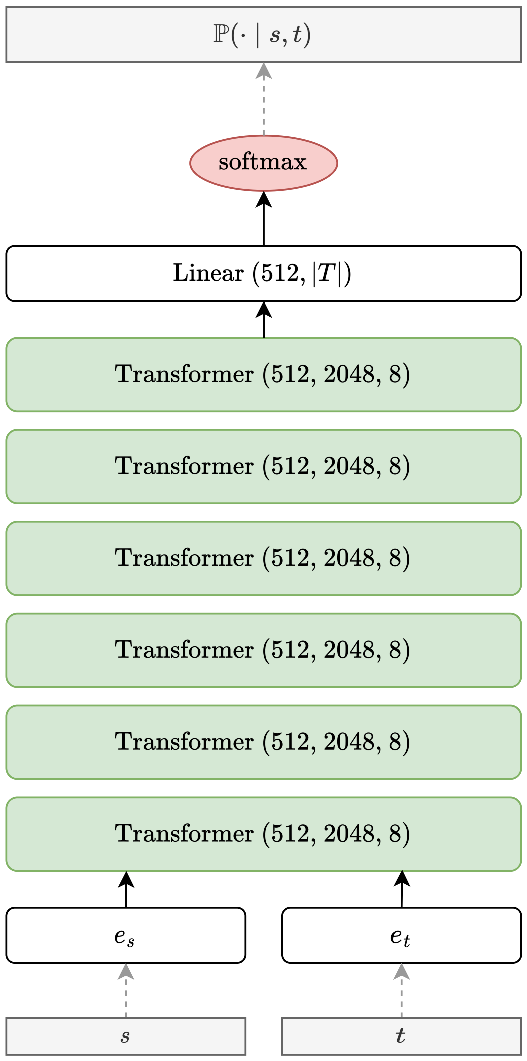

$\mathbb{P}(\cdot \mid s, t)$
softmax
Linear (512, $|T|$)
Transformer (512, 2048, 8)
Transformer (512, 2048, 8)
Transformer (512, 2048, 8)
Transformer (512, 2048, 8)
Transformer (512, 2048, 8)
Transformer (512, 2048, 8)
e_s
e_t
s
t

Figure B.1: Model architecture. Transformer layers (green) are frozen during the fine-tuning phase. Embeddings and linear projection (white) are learned during fine-tuning.

Index of Work