Lexicon Learning for Few-Shot Neural Sequence Modeling

Anonymous ACL-IJCNLP submission

Abstract

Sequence-to-sequence transduction is the core problem in language processing applications as diverse as semantic parsing, machine translation, and instruction following. The neural network models that provide the dominant solution to these problems are brittle, especially in low-resource settings: they fail to generalize correctly or systematically from small datasets. Past work has shown that many failures of systematic generalization arise from neural models' inability to disentangle *lexical* phenomena from *syntactic* ones. To address this, we augment neural decoders with a *lexical translation mechanism* that generalizes existing copy mechanisms to incorporate learned, decontextualized, token-level translation rules. We describe how to initialize this mechanism using a variety of lexicon learning algorithms, and show that it improves systematic generalization on a diverse set of sequence modeling tasks drawn from cognitive science, logical semantics, and machine translation.

1 Introduction

Humans exhibit a set of *structured* and remarkably *consistent* inductive biases when learning from language data. For example, in both natural language acquisition and toy language-learning problems like the one depicted in Table 1, human learners exhibit a preference for systematic and compositional interpretation rules (Guasti 2017, Chapter 4; Lake et al. 2019). These inductive biases in turn support behaviors like one-shot learning of new concepts (Carey and Bartlett, 1978). But in natural language processing, recent work has found that state-of-the-art neural models, while highly effective at in-domain prediction, fail to generalize in human-like ways when faced with rare phenomena and small datasets (Lake and Baroni, 2018) posing a fundamental challenge for NLP tools in the low-data regime.

TRAIN	TEST
dax: r, lug: b, wif: g, zup: y	zup fep: y y y
lug fep: b b b	zup blicket lug: ___?
dax fep: r r r	dax blicket zup: ___?
lug blicket wif: b g b	zup kiki dax: __?
wif blicket dax: g r b	wif kiki zup: __?
lug kiki wif: g b	
dax kiki lug: b r	

Table 1: A part of the Colors dataset from Lake et al. (2019), a simple sequence-to-sequence translation task. The output vocabulary is only the colored circles r, g, b, y. Humans can reliably fill in the missing test labels on the basis of a small training set, but standard neural models cannot.

Pause for a moment to fill in the missing labels in Table 1. While doing so, which training examples did you pay the most attention to? How many times did you find yourself saying *means* or *maps to*? Explicit representations of lexical items and their meanings play a key role diverse models of syntax and semantics (Joshi and Schabes, 1997; Pollard and Sag, 1994; Bresnan et al., 2015). But one of the main findings in existing work on generalization in *neural* models is that they fail to cleanly separate **lexical** phenomena from **syntactic** ones (Lake and Baroni, 2018). Given a dataset like the one depicted in Table 1, models conflate (lexical) information about the correspondence between *zup* and y with the (syntactic) fact that y appears only in a sequence of length 1 at training time. Longer input sequences containing the word *zup* in new syntactic contexts cause models to output tokens only seen in longer sequences (Section 5).

In this paper, we introduce a new output layer for neural sequence models that facilitates (but does not enforce) the learning of context-independent word meanings. We augment decoders with a **lexical translation mechanism** which generalizes neural copy mechanisms (e.g. See et al., 2017) and enables models to generate token-level translations purely attentionally. While the lexical translation mechanism is quite general, we focus here on its ability to improve few-shot learning in sequence-to-sequence models. On a suite of challenging tests of few-shot semantic parsing and instruction following our model exhibits strong generalization, achieving the highest reported results for neural sequence models on datasets as diverse as COGS (Kim and Linzen 2020, with 24155 training examples) and Colors (Lake et al. 2019, with 14). Our approach also generalizes to real-world tests of few-shot learning, improving BLEU scores (Papineni et al., 2002) by 1.2 on a low-resource English–Chinese machine translation task (2.2 on test sentences requiring one-shot word learning).

In an additional set of experiments, we explore effective procedures for initializing the lexical translation mechanism. We introduce a simple, logical lexicon learning rule that serves as an effective initializer, performing comparably to (and sometimes better than) existing lexicon learning algorithms derived from information theory, statistical machine translation, and Bayesian cognitive modeling. We then explore joint learning of the lexicon and decoder, but find (surprisingly) that this gives only marginal improvements over a fixed initialization of the lexicon. In summary, this work[1]:

- Introduces a new, lexicon-based output mechanism for neural encoder–decoder models.
- Proposes and investigates procedures for initializing the parameters of this mechanism.
- Uses it to solve challenging tests of generalization in instruction following, semantic parsing and machine translation.

A great deal of past work has suggested that neural models come equipped with an inductive bias that makes them fundamentally ill-suited to human-like generalization about language data, especially in the low-data regime (e.g. Fodor et al., 1988; Marcus, 2018). Our results indicate that this is not the case, and that observed problems stem from a comparatively simple conflation of syntax and the lexicon. By offloading the easier lexicon learning problem to simpler models, neural sequence models are actually quite effective at modeling (and generalizing about) about syntax.

2 Related Work

Systematic generalization in neural sequence models The desired inductive biases noted above are usually grouped together as "systematicity" but in fact involve a variety of phenomena: one-shot learning of new concepts and composition rules (Lake and Baroni, 2018), zero-shot interpretation of novel words from context cues (Gandhi and Lake, 2020), and interpretation of known concepts in novel syntactic configurations (Keysers et al., 2020; Kim and Linzen, 2020). What they share is a common expectation that learners should associate specific production or transformation rules with specific input tokens (or phrases), and generalize to use of these tokens in new contexts.

Recent years have seen tremendous amount of modeling work aimed at encouraging these generalizations in neural models, primarily by equipping them with symbolic scaffolding in the form of program synthesis engines (Nye et al., 2020), stack machines (Grefenstette et al., 2015; Liu et al., 2020), or symbolic data transformation rules (Gordon et al., 2019; Andreas, 2020). A parallel line of work has investigated the role of continuous representations in systematic generalization, proposing improved methods for pretraining (Furrer et al., 2020) and procedures for removing irrelevant contextual information from word representations (Arthur et al., 2016; Russin et al., 2019; Thrush, 2020). The latter two approaches proceed from very similar intuition to ours, aiming to diesntangle word meanings from syntax in encoder representations via alternative attention mechanisms and adversarial training. Our approach instead focuses on providing an explicit lexicon to the decoder; as discussed below, this appears to be considerably more effective.

Copying and lexicon learning In neural encoder–decoder models, the clearest example of benefits from special treatment of word-level production rules is the *copy mechanism*. A great deal of past work has found that neural models benefit from learning a structural copy operation that selects output tokens directly from the input sequence without requiring token identity to be carried through all neural computation in the encoder and the decoder. These mechanisms are described in detail in Section 3, and are widely used in models for language generation, summarization and semantic parsing. Our work generalizes these models to structural operations on the input that replace copying with general context-independent token-level translation.

As will be discussed, the core of our approach

[1]We will release our code base after the anonymity period.

Inputs	Outputs	Lexicon Entries
A crocodile blessed William .	`crocodile(x_1) AND bless.agent (x_2, x_1) AND bless.theme (x_2, William)`	*blessed* $\mapsto$ `bless`
William needed to walk .	`need.agent (x_1 , William) AND need.xcomp(x_1, x_3) AND walk.agent (x_3, William)`	*needed* $\mapsto$ `need`
		William $\mapsto$ `William`
Many moons orbit around Saturn	許多 衛星 繞著 土星 運行.	*saturn* $\mapsto$ 土星
Earth is a planet .	地球 是 一個 行星.	*earth* $\mapsto$ 地球
		moon $\mapsto$ 衛星
walk around left	`LTURN IWALK LTURN IWALK LTURN IWALK LTURN IWALK`	*walk* $\mapsto$ `IWALK`
turn right	`RTURN`	*jump* $\mapsto$ `IJUMP`
turn left	`LTURN`	*right* $\mapsto$ `RTURN`
jump	`IJUMP`	*left* $\mapsto$ `LTURN`
jump opposite right after look left	`LTURN ILOOK RTRUN IJUMP RTURN IJUMP`	*look* $\mapsto$ `ILOOK`

Table 2: We present example (input,output) pairs from COGS, English-to-Chinese machine translation and SCAN datasets. We also present some of the lexicon entries which can be learned by proposed lexicon learning methods and that are helpful to make generalizations required in each of the datasets.

is a (non-contextual) lexicon that maps individual input tokens to individual output tokens. Learning lexicons like this is of interest in a number of communities in NLP and language science more broadly. A pair of representative approaches (Brown et al., 1993; Frank et al., 2007) will be discussed in detail below; other work on lexicon learning for semantics and translation includes Liang et al. (2009); Goldwater (2007); Haghighi et al. (2008) among numerous others.

Finally, and closest to the modeling contribution in this work, several previous papers have proposed alternative generalized copy mechanisms for tasks other than semantic lexicon learning. Concurrent work by Prabhu and Kann (2020) introduces a similar approach for grapheme-to-phoneme translation (with a fixed functional lexicon rather than a trainable parameter matrix), and Nguyen and Chiang (2018) describe a less expressive mechanism without a token-level choice between the lexical translation mechanism and ordinary decoder output. Akyürek et al. (2020) describes a model in which a copy mechanism is combined with a retrieval-based generative model; like the present work, that model effectively disentangles syntactic and lexical information by using training examples as implicit representations of lexical correspondences.

We generalize and extend this previous work in a number of ways, providing a new parameterization of attentive translation and a detailed study of initialization and training. But perhaps the most important contribution of this work is the observation that many of the hard problems studied as "compositional generalization" have direct analogues in more conventional NLP problems, especially machine translation. Research on systematicity and generalization would benefit from closer attention to the ingredients of effective translation at scale.

3 Sequence-to-Sequence Models With Lexical Translation Mechanisms

This paper focuses on sequence-to-sequence **language understanding** problems like the ones depicted in Table 2, in which the goal is to map from a natural language **input** $x = [x_1, x_2, \ldots, x_n]$ to a structured **output** $y = [y_1, y_2, \ldots, y_m]$—a logical form, action sequence, or translation. We assume input tokens x_i are drawn from a **input vocabulary** $\mathcal{V}_x$, and output tokens from a corresponding **output vocabulary** $\mathcal{V}_y$.

Neural encoder–decoders Our approach builds on the standard neural encoder–decoder model with attention (Bahdanau et al., 2015). In this model, an **encoder** represents the input sequence $[x_1, \ldots, x_n]$ as a sequence of representations $[e_1, \ldots, e_n]$

$$e = \mathtt{encoder}(x) \tag{1}$$

Next, a **decoder** generates a distribution over output sequences y according to the sequentially:

$$\log p(y \mid x) = \sum_i \log p(y_i \mid y_{<i}, e, x) \tag{2}$$

Here we specifically consider decoders with **attention.**[2]. When predicting each output token y_i, we assign each input token an **attention weight** α_i^j as in Eq. (3). Then, we construct a context representation c_i as the weighted sum of encoder representations e_i:

$$\alpha_i^j \propto \exp(h_i^\top W_{att}\, e_j) \tag{3}$$

$$c_i = \sum_j \alpha_i^j e_j \tag{4}$$

[2]All experiments in this paper use LSTM encoders and decoders, but it could be easily integrated with CNNs or transformers; Gehring et al. 2017; Vaswani et al. 2017). We only assume access to a final layer h_i, and final attention weights α_is; their implementation does not matter.

The output distribution over $\mathcal{V}_y$, which we denote $p_{\text{write},i}$, is calculated by a final projection layer:

$$p(y_i = w|x) = p_{\text{write}_i}(w) \propto \exp(W_{\text{write}}[c_i, h_i]) \quad (5)$$

Copying A popular extension of the model described above is the **copy mechanism**, in which output tokens can be copied from the input sequence in addition to being generated directly by the decoder (Jia and Liang, 2016; See et al., 2017). Using the decoder hidden state h_i from above, the model first computes a **gate probability**:

$$p_{\text{gate}} = \sigma(w_{\text{gate}}^\top h_i) \quad (6)$$

and then uses this probability to interpolate between the distribution in Eq. (5) and a **copy distribution** that assigns to each word in the output vocabulary a probability proportional to that word's weight in the attention vector over the input:

$$p_{\text{copy}}(y_i = w \mid x) = \sum_j \mathbb{1}[x_j = w] \cdot \alpha_i^j \quad (7)$$

$$\begin{aligned} p(y_i = w \mid x) &= p_{\text{gate}} \cdot p_{\text{write}}(y_i = w \mid x) \\ &+ (1 - p_{\text{gate}}) \cdot p_{\text{copy}}(y_i = w \mid x) \quad (8) \end{aligned}$$

(note that this implies $\mathcal{V}_y \supseteq \mathcal{V}_x$).

Content-independent copying is particularly useful in tasks like summarization and machine translation where rare words (like names) are often reused between the input and output.

Our model: Lexical translation The **lexical translation mechanism** we introduce in this work is similar to the copy mechanism, but introduces an additional layer of indirection between the input sequence x and the output prediction y_i as shown in Fig. 1. Specifically, after selecting an input token $x_j \in \mathcal{V}_x$, the decoder can "translate" it to a context-independent output token $\in \mathcal{V}_y$ prior to the final prediction. We equip the model with an additional **lexicon parameter** L, a $|\mathcal{V}_x| \times |\mathcal{V}_y|$ matrix in which $\sum_j L_{ij} = 1$, and finally define

$$p_{\text{lex}}(y_i = w \mid x) = \sum_j L_{x_j,w} \cdot \alpha_i^j \quad (9)$$

$$\begin{aligned} p(y_i = w \mid x) &= p_{\text{gate}} \cdot p_{\text{write}}(y_i = w \mid x) \\ &+ (1 - p_{\text{gate}}) \cdot p_{\text{lex}}(y_i = w \mid x) \quad (10) \end{aligned}$$

Note that when $\mathcal{V}_x = \mathcal{V}_y$ and $L = I$ is diagonal, this is identical to the original copy mechanism. However, this approach can in general be used to produce a much larger set of tokens. As shown in Table 2, coherent token-level translation rules can be identified for many tasks; the lexical translation mechanism allows them to be stored explicitly, using parameters of the base sequence-to-sequence model to record general structural behavior and more complex, context-dependent translation rules.

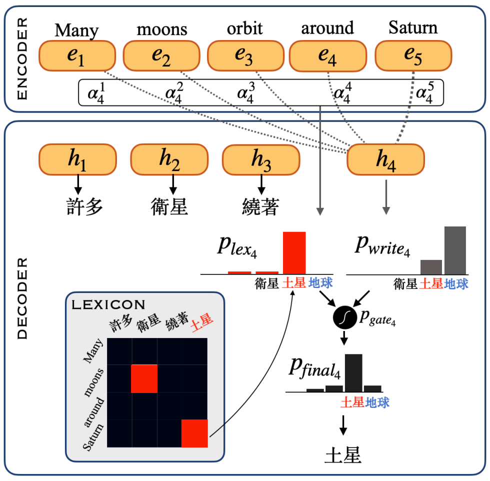

Figure 1: An encoder-decoder model with a lexical translation mechanism applied to English-to-Chinese translation. At decoder step $t = 4$, attention is focused on the English token *Saturn*. The lexical translation mechanism is activated by p_{gate}, and the model outputs the token 土星 directly from the lexicon. 地球 means *Earth* that is a frequent than *Saturn* in the training set.

4 Initializing the Lexicon

The lexicon parameter L in the preceding section can be viewed as an ordinary fully-connected layer inside the copy mechanism, and trained end-to-end with the rest of the network. As with other neural network parameters, however, our experiments will show that the initialization of the parameter L significantly impacts downstream model performance, and specifically benefits from initialization with a set of input–output mappings learned with an offline lexicon learning step. Indeed, while not widely used in neural sequence models (though c.f. Section 2), lexicon-based initialization was a standard feature of many complex non-neural sequence transduction models, including semantic parsers (Kwiatkowski et al., 2011) and phrase-based machine translation systems (Koehn et al., 2003).

But an important distinction between our approach and these others is the fact that we can handle outputs that are not (transparently) compositional. Not every fragement of an input will correspond to a fragment of an ouput: for example, *thrice* in SCAN has no corresponding output token and instead describes a structural transformation.

Moreover, the lexicon is not the only way to generate: complex mappings can also be learned by p_{write} without going through the lexicon at all.

Thus, while most existing work on lexicon learning aims for complete coverage of all word meanings, the model described in Section 3 benefits from a lexicon with *high-precision* coverage of *rare phenomena* that will be hard to learn in a normal neural model. Lexicon learning is widely studied in language processing and cognitive modeling, and several approaches with very different inductive biases exist. To determine how to best initialize L, we begin by reviewing three algorithms in Section 4.1, and identify ways in which each of them fail to satisfy the high precision criterion above. In Section 4.2, we introduce a simple new lexicon learning rule that addresses this shortcoming.

4.1 Existing Approaches to Lexicon Learning

Statistical alignment In the natural language processing literature, the so-called IBM translation models (Brown et al., 1993) have served as some of the most popular procedures for learning token-level input–output mappings. While originally developed for machine translation they have also been used to initialize semantic lexicons for semantic parsing (Kwiatkowski et al., 2011) and grapheme-to-phoneme conversion (Rama et al., 2009). We initialize the lexicon parameter L using Model 2.

Model 2 defines a generative process in which source words y_i are generated from target words x_j via latent alignments a_i. Specifically, given a (source, target) pair with n source words and m target words, the probability that the target word i is aligned to the source word j is:

$$p(a_i = j) \propto \exp\left(-\left|\frac{i}{m} - \frac{j}{n}\right|\right) \tag{11}$$

Finally, each target word is generated by its aligned source word via a parameter θ: $p(y_i = w) = \theta(v, x_{a_i})$. Alignments a_i and lexical parameters θ can be jointly estimated using the expectation–maximization algorithm (Dempster et al., 1977).

In neural models, rather than initializing lexical parameters L directly with corresponding IBM model parameters θ, we run Model 2 in both the forward and reverse directions, then extract counts by *intersecting* these alignments and applying a softmax with temperature τ:

$$L_{vw} \propto \exp\left(\tau^{-1} \sum_{(x,y)} \sum_i \mathbb{1}[x_{a_i} = v]\mathbb{1}[y_i = w]\right) \tag{12}$$

For all lexicon methods discussed in this paper, if an input v is not aligned to any output w, we map it to itself if $\mathcal{V}_x \subseteq \mathcal{V}_y$. Otherwise we align it uniformly to any unmapped output words (a *mutual exclusivity bias*, Gandhi and Lake 2020).

Mutual information Another, even simpler procedure for building a lexicon is based on identifying pairs that have high *pointwise mutual information*. We estimate this quantity directly from co-occurrence statistics in the training corpus:

$$\text{pmi}(v; w) = \log \frac{\#(v, w)}{\#(v)\#(w)} + \log |D_{train}| \tag{13}$$

where $\#(w)$ is the number of times the word w appears in the training corpus and $\#(w, v)$ is the number of times that w appears in the input and v appears in the output. Finally, we populate the parameter L via a softmax transformation: $L_{vw} \propto \exp((1/\tau)\ \text{pmi}\,(v; w))$.

Bayesian lexicon learning Last, we explore the Bayesian cognitive model of lexicon learning described by Frank et al. (2007). Like IBM model 2, this model is defined by a generative process; here, however, the lexicon itself is part of the generative model. A lexicon ℓ is an (unweighted, many-to-many) map defined by a collection of pairs (x, y) with a *description length* prior: $p(\ell) \propto e^{-|\ell|}$ (where $|\ell|$ is the number of (input, output) pairs in the lexicon). As in Model 2, given a meaning y and a natural-language description x, each x_i is generated independently. We define the probability of a word being used *non-referentially* as $p_{\text{NR}}(x_i \mid \ell) \propto 1$ if $x_i \notin \ell$ and κ otherwise. The probability of being used referentially is: $p_{\text{R}}(x_i \mid y_j, \ell) \propto \mathbb{1}_{(x_i, y_j) \in \ell}$. Finally,

$$\begin{aligned} p(x_i \mid y_j, \ell) &= (1-\gamma) p_{\text{NR}}(x_i \mid \ell) \\ &\quad + \gamma \sum_j p_{\text{R}}(x_i \mid y_j, \ell) \end{aligned} \tag{14}$$

To produce a final lexical translation matrix L for use in our experiments, we set $L_{vw} \propto \exp((1/\tau)\ p((v, w) \in \ell))$: each entry in L is the posterior probability that the given entry appears in a lexicon under the generative model above. This quantity is computed using the Metropolis–Hastings algorithm, with details described in Appendix C.

4.2 A Simpler Lexicon Learning Rule

Example lexicons learned by the three models above are depicted in Fig. 2 for the SCAN task

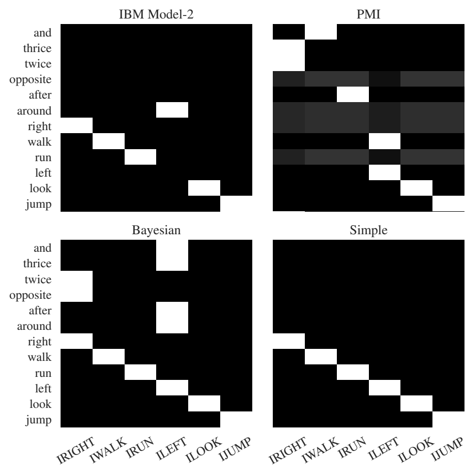

Figure 2: Learned lexicons for the *around right* split in SCAN ($\tau = 0.1$). Simple lexicon produces correct alignments, while other methods fail due to correlation between 'around' and 'left' in training data.

shown in Table 2. Lexicons learned for remaining tasks can be found in Appendix B. It can be seen that all three models produce errors: the PMI and Bayesian lexicons contain too many entries (in both cases, numbers are associated with the turn right action and prepositions are associated with the turn left action). For the IBM model, one of the alignments is confident but wrong, because the *around* preposition is associated with turn left action. Motivated by these errors, and by the aforementioned demand for high-precision coverage of rare phenomena in our use case, we describe a simple logical procedure for extracting lexicon entries that, surprisingly, outperforms all three baseline methods in most of our experiments.

What makes an effective, precise lexicon learning rule? As a first step, consider a maximally restrictive criterion (which we'll call C_1) that extracts only pairs (v, w) for which the presence of v in the input is a *necessary and sufficient condition* for the presence of w in the output.

$$\textit{nec.}(v, w) = \forall xy.\ (w \in y) \rightarrow (v \in x) \qquad (15)$$

$$\textit{suff.}(v, w) = \forall xy.\ (v \in x) \rightarrow (w \in y) \qquad (16)$$

$$C_1(v, w) = \textit{nec.}(v, w) \wedge \textit{suff.}(v, w) \qquad (17)$$

C_1 is too restrictive: in many language understanding problems, the mapping from *surface forms* to meanings is many-to-one (in Table 2, both *blessed* and *bless* are associated with the logical form `bless`). Such mappings cannot be learned by the algorithm described above. We can relax the necessity condition slightly, requiring *either* that v is a necessary condition for w, or is part of a group that collectively explains all occurrences of w:

$$\textit{no-winner}(w) = \nexists v'.\ C_1(v', w) \qquad (18)$$

$$C_2(v, w) = \textit{suff.}(v, w) \wedge (\textit{nec.}(v, w) \vee \textit{no-win.}(w)) \qquad (19)$$

As a final refinement, we note that C_2 is likely to capture function words that are present in most sentences, and exclude these by restricting the lexicon to words below a certain frequency threshold:

$$C_3 = C_2 \wedge \left|\{v' : \textit{suff.}(v', w)\}\right| \leq \epsilon \qquad (20)$$

The lexicon matrix L is computed by taking the word co-occurrence matrix, zeroing out all entries where C_3 doesn't hold, then computing a softmax: $L_{vw} \propto C_3(v, w) \exp((1/\tau)\ \#(v, w))$. Surprisingly, as shown in Fig. 2 and and evaluated below, this rule (which we label Simple) produces the most effective lexicon initializer for three of the four tasks we study. The simplicity (and extreme conservativity) of this rule highlight the different demands on L made by our model and more conventional (e.g. machine translation) approaches: the lexical translation mechanism benefits from even a small number of precise mappings.

5 Experiments

We investigate the effectiveness of the lexical translation mechanism on sequence-to-sequence models four tasks, three focused on compositional generalization and one on low-resource machine translation. In all experiments, we use an LSTM encoder–decoder with attention as the base predictor. We compare our approach (and variants) with two other baselines: GECA (Andreas, 2020) (a data augmentation scheme, see Section 2) and SynAtt (Russin et al., 2019) (see Section 2). Hyper-parameter selection details are given in the Appendix C. Unless otherwise stated, we use $\tau = 0$ and do not fine-tune L after initialization.

5.1 Colors

Task The Colors sequence translation task (see Appendix A for full dataset) was developed to measure *human* inductive biases in sequence-to-sequence learning problems. It poses an extreme test of low-resource learning for neural sequence models: it has only 14 training examples that combine four named colors and three composition operations that perform concatenation, repitition and

	COGS	around right (SCAN)	jump (SCAN)	Colors
LSTM	0.51 ±0.05	0.09 ±0.05	0.00 ±0.00	0.00 ±0.00
GECA	0.48 ±0.05	**0.98** ±0.02	**1.00** ±0.00	0.41 ±0.11
SyntAtt	0.15 ±0.14	0.28 ±0.26	0.57 ±0.38	0.57 ±0.26
LSTM + copy	0.66 ±0.03	-	-	-
LSTM + Lex.: Simple	**0.82** ±0.01	0.95 ±0.01	0.92 ±0.17	**0.78** ±0.04
LSTM + Lex.: PMI	**0.82** ±0.01	0.02 ±0.04	0.95 ±0.08	0.49 ±0.11
LSTM + Lex.: IBMM2	**0.82** ±0.00	0.00 ±0.00	0.92 ±0.17	**0.78** ±0.04
LSTM + Lex.: Bayesian	0.70 ±0.04	0.02 ±0.04	0.82 ±0.21	0.53 ±0.14

Table 3: Exact match accuracy results for baselines and lexicon learning models on 4 different compositional generalization splits. Errors are standard deviation among 10 different seeds. Unbolded numbers are significantly($p < 0.01$) worse than the best result in the column. Models with lexical translation mechanisms and Simple initialization consistently improve over ordinary LSTMs.

wrapping. Liu et al. (2020) solve this dataset with a symbolic stack machine; to the best of our knowledge, our approach is the first "pure" neural sequence model to obtain non-trivial accuracy.

Results Both the Simple and IBMM2 initializers produce a lexicon that maps only color words to colors. Both, combined with the lexical translation mechanism, obtain an average test accuracy of 78% across 20 runs, nearly matching the human accuracy of 81% reported by Lake et al. (2019). The two test examples most frequently predicted incorrectly require generalization to longer sequences than seen during training. More details (including example-level model and human accuracies) are presented in the appendix Appendix A). These results show that LSTMs are quite effective at learning systematic sequence transformation rules from $\approx$ 3 examples per function word when equipped with lexical translations. Generalization to longer sequences remains as an important challenge for future work.

5.2 SCAN

Task SCAN (Lake and Baroni, 2018) is a larger collection of tests of systematic generalization that pair synthetic English commands (e.g. *turn left twice and jump*) to action sequences (e.g. `LTURN LTURN IJUMP`) as shown in Table 2. Following previous work, we focus on the *jump* and *around right* splits, each of which features roughly 15,000 training examples, and evaluate models' ability to perform 1-shot learning of new primitives (*jump*) and zero-shot interpretation of composition rules (*around right*). While these tasks are now solved by a number of specialized approaches, they remain a challenge for conventional neural sequence models, and an important benchmark for new models.

Results In the *jump* split, all initializers improve significantly over the base LSTM when combined with lexical translation. Most methods achieve 99% accuracy at least once across seeds. These results are slightly behind GECA (in which all runs succeed) but ahead of SynAtt.[3] Again, they show that lexicon learning is effective for systematic generalization, and that simple initializers (PMI and Simple) outperform complex ones.

5.3 COGS

Task COGS (Compositional Generalization for Semantic Parsing; Kim and Linzen 2020) is an automatically generated English-language semantic parsing dataset that tests systematic generalization in learning language-to-logical-form mappings. It includes 24155 training examples. Compared to the Colors and SCAN datasets, it has a larger vocabulary (876 tokens) and finer-grained inventory of syntactic generalization tests (Table 6).

Results Notably, because some tokens appear in both inputs and logical forms in the COGS task, even a standard sequence-to-sequence model with copying significantly outperforms the baseline models in the original work of Kim and Linzen (2020), solving most tests of generalization over syntactic roles for nouns (but performing worse at generalizations over verbs, including passive and dative alternations). As above, the lexical translation mechanism (with any of the proposed initializers) provides further improvements, mostly for verbs that baselines model incorrectly (Table 6).

5.4 Machine Translation

Task To demonstrate that this approach is useful beyond synthetic tests of generalization, we eval-

[3]SynAtt results here are lower than reported in the original paper, which discarded runs with a test accuracy of 0%.

	ENG-CHN	
	full	1-shot
LSTM	24.18 ± 0.37	17.47 ± 0.64
LSTM + Lex.: Simple	24.35 ± 0.09	18.46 ± 0.19
LSTM + Lex.: IBMM2	**25.49** ± 0.42	**19.62** ± 0.64

Table 4: BLEU scores for English-Chinese translation dataset. *full* shows results on the full test set, and *1-shot* shows results for text examples in which the English text contains a token seen only once during training.

	Accuracy
LSTM	0.51 ± 0.06
└ Lex.: Uniform	0.56 ± 0.07
└ Lex.: Simple	0.82 ± 0.01
└ Soft	0.82 ± 0.01
└ Learned	**0.83** ± 0.01

Table 5: Ablation experiments on the COGS dataset. *Uniform* shows results for a lexicon initialized to a uniform distribution. *Soft* sets $\tau = 0.1$ with the Simple lexicon learning rule (rather than 0 in previous experiments). *Learned* shows results for a soft lexicon fine-tuned during training. Learning improves significantly ($p < 0.01$) but very slightly over fixed initialization.

uate it on a low-resource English–Chinese translation task. (The Tatoeba[4] dataset processed by Kelly 2021). For our experiments, we split the data randomly into 19222 training and 2402 test pairs.

Results Results are shown in Table 4. Models with a lexical translation mechanism initialized with Simple or IBMM2 obtain modest improvements (up to 1.5 BLEU) over the baseline. Notably, if we restrict evaluation to test sentences featuring English words that appeared only once in the training set, BLEU improves by more than 2 points, demonstrating that this approach is particularly effective at one-shot word learning (or *fast mapping*, Carey and Bartlett 1978). Fig. 1 shows an example from this dataset, in which the model learns to reliably translate *Saturn* from a single training example. These experiments show that the lexical translation mechanism is effective in natural tasks with large vocabularies and complex grammars.

5.5 Fine-Tuning the Lexicon

In all the experiments above, the lexicon was discretized ($\tau = 0$) and frozen prior to training. In this final section, we revisit that decision, evaluating whether the parameter L can be learned from scratch, or effectively fine-tuned along with decoder parameters. Experiments in this section

[4] https://tatoeba.org/

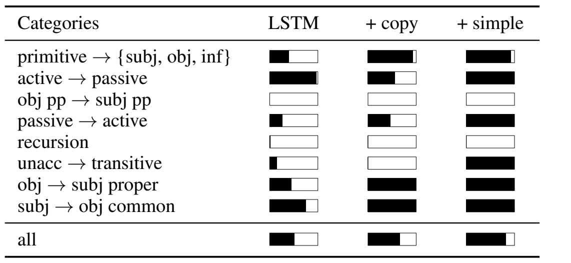

Table 6: COGS accuracy breakdown according to syntactic generalization types for word usages. The label $a \rightarrow b$ indicates that syntactic context a appears in the training set and b in the test set.

focus on the COGS dataset.

Offline initialization of the lexicon is crucial. Rather than initializing L using any of the algorithms described in Section 3, we initialized L to a uniform distribution for each word and optimized during training. This improves over the base LSTM (*Uniform* in Table 5), but performs significantly worse than pre-learned lexicons.

Benefits from fine-tuning are minimal. We first increased the temperature parameter τ to 0.1 (providing a "soft" lexicon); this on its own gave the same results as described above (Table 5. *Soft*). Finally, we updated this soft initialization via gradient descent; this provided only a 1% improvement on COGS (Table 5, *Learned*). One important feature of COGS (and other tests of compositional generalization) is perfect *training* accuracy is easily achieved; thus, there is little pressure on models to learn generalizable lexicons. This pressure must instead come from inductive bias in the initializer.

6 Conclusion

We have described a *lexical translation mechanism* for representing token-level translation rules in neural sequence models. We have additionally described a simple initialization scheme for this lexicon that outperforms a variety of existing algorithms. Together, lexical translation and proper initialization enable neural sequence models to solve a diverse set of tasks—including semantic parsing and machine translation—that require 1-shot word learning and 0-shot compositional generalization. Future work might focus on generalization to longer sequences, learning of atomic but non-concatenative translation rules, and online lexicon learning in situated contexts.

References

Ekin Akyürek, Afra Feyza Akyürek, and Jacob Andreas. 2020. Learning to recombine and resample data for compositional generalization. *arXiv preprint arXiv:2010.03706*.

Jacob Andreas. 2020. Good-enough compositional data augmentation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7556–7566.

Philip Arthur, Graham Neubig, and Satoshi Nakamura. 2016. Incorporating discrete translation lexicons into neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1557–1567.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

Joan Bresnan, Ash Asudeh, Ida Toivonen, and Stephen Wechsler. 2015. *Lexical-functional syntax*. John Wiley & Sons.

Peter F Brown, Stephen A Della Pietra, Vincent J Della Pietra, and Robert L Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311.

Susan Carey and Elsa Bartlett. 1978. Acquiring a single new word. *Papers and Reports on Child Language Development*, 2.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. 1977. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22.

Chris Dyer, Victor Chahuneau, and Noah A Smith. 2013. A simple, fast, and effective reparameterization of ibm model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 644–648.

Jerry A Fodor, Zenon W Pylyshyn, et al. 1988. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1-2):3–71.

Michael C. Frank, Noah D. Goodman, and J. Tenenbaum. 2007. A bayesian framework for cross-situational word-learning. In *NeurIPS*.

Daniel Furrer, Marc van Zee, Nathan Scales, and Nathanael Schärli. 2020. Compositional generalization in semantic parsing: Pre-training vs. specialized architectures. *arXiv preprint arXiv:2007.08970*.

Kanishk Gandhi and Brenden M Lake. 2020. Mutual exclusivity as a challenge for deep neural networks. *Advances in Neural Information Processing Systems*, 33.

Jonas Gehring, M. Auli, David Grangier, Denis Yarats, and Yann Dauphin. 2017. Convolutional sequence to sequence learning. In *ICML*.

Sharon J Goldwater. 2007. *Nonparametric Bayesian Models of Lexican Acquisition*. Citeseer.

Jonathan Gordon, David Lopez-Paz, Marco Baroni, and Diane Bouchacourt. 2019. Permutation equivariant models for compositional generalization in language. In *International Conference on Learning Representations*.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. 2015. Learning to transduce with unbounded memory. In *NIPS*.

Maria Teresa Guasti. 2017. *Language acquisition: The growth of grammar*. MIT press.

A. Haghighi, Percy Liang, Taylor Berg-Kirkpatrick, and D. Klein. 2008. Learning bilingual lexicons from monolingual corpora. In *ACL*.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. *arXiv preprint arXiv:1606.03622*.

Aravind K Joshi and Yves Schabes. 1997. Tree-adjoining grammars. In *Handbook of formal languages*, pages 69–123. Springer.

Charles Kelly. 2021. [link].

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, Dmitry Tsarkov, Xiao Wang, Marc van Zee, and Olivier Bousquet. 2020. Measuring compositional generalization: A comprehensive method on realistic data. In *ICLR*.

Najoung Kim and Tal Linzen. 2020. Cogs: A compositional generalization challenge based on semantic interpretation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9087–9105.

Philipp Koehn, F. Och, and D. Marcu. 2003. Statistical phrase-based translation. In *HLT-NAACL*.

T. Kwiatkowski, Luke Zettlemoyer, S. Goldwater, and Mark Steedman. 2011. Lexical generalization in ccg grammar induction for semantic parsing. In *EMNLP*.

Brenden Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning*, pages 2873–2882. PMLR.

Brenden M Lake, Tal Linzen, and Marco Baroni. 2019. Human few-shot learning of compositional instructions. *arXiv preprint arXiv:1901.04587*.

Percy Liang, Michael I Jordan, and Dan Klein. 2009. Learning semantic correspondences with less supervision. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 91–99.

Qian Liu, Shengnan An, Jian-Guang Lou, Bei Chen, Zeqi Lin, Yan Gao, Bin Zhou, Nanning Zheng, and Dongmei Zhang. 2020. Compositional generalization by learning analytical expressions. *Advances in Neural Information Processing Systems*, 33.

Gary Marcus. 2018. Deep learning: A critical appraisal. *arXiv preprint arXiv:1801.00631*.

Toan Q. Nguyen and David Chiang. 2018. Improving lexical choice in neural machine translation. *ArXiv*, abs/1710.01329.

Maxwell I Nye, Armando Solar-Lezama, Joshua B Tenenbaum, and Brenden M Lake. 2020. Learning compositional rules via neural program synthesis. *arXiv preprint arXiv:2003.05562*.

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

Carl Pollard and Ivan A Sag. 1994. *Head-driven phrase structure grammar*. University of Chicago Press.

Nikhil Prabhu and K. Kann. 2020. Making a point: Pointer-generator transformers for disjoint vocabularies. In *AACL*.

Taraka Rama, Anil Kumar Singh, and Sudheer Kolachina. 2009. Modeling letter-to-phoneme conversion as a phrase based statistical machine translation problem with minimum error rate training. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Student Research Workshop and Doctoral Consortium*, pages 90–95.

Jake Russin, Jason Jo, Randall C O'Reilly, and Yoshua Bengio. 2019. Compositional generalization in a deep seq2seq model by separating syntax and semantics. *arXiv preprint arXiv:1904.09708*.

Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 1073–1083.

Tristan Thrush. 2020. Compositional neural machine translation by removing the lexicon from syntax. *arXiv preprint arXiv:2002.08899*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, L. Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *ArXiv*, abs/1706.03762.

A Colors Dataset & Detailed Results

Here we present the full dataset in Table 7 from Lake et al. (2019), and detailed comparison of each model and human results in Table 8.

TRAIN		TEST	
INPUT	OUTPUT	INPUT	OUTPUT
dax	r	zup fep	y y y
lug	b	zup kiki dax	r y
wif	g	wif kiki zup	y g
zup	y	zup blicket lug	y b y
lug fep	b b b	dax blicket zup	r y r
dax fep	r r r	wif kiki zup fep	y y y g
lug blicket wif	b g b	zup fep kiki lug	b y y y
wif blicket dax	g r g	lug kiki wif blicket zup	g y g b
lug kiki wif	g b	zup blicket wif kiki dax fep	r r r y g y
dax kiki lug	b r	zup blicket zup kiki zup fep	y y y y y y
lug fep kiki wif	g b b b		
wif kiki dax blicket lug	r b r g		
lug kiki wif fep	g g g b		
wif blicket dax kiki lug	b g r g		

Table 7: Full Colors dataset with Train and Test examples (Lake et al., 2019)

Test Examples	Simple/IBM-M2	Bayesian	GECA	SyntAtt	Human
zup fep	1.00 ±0.000	0.95 ±0.22	1.00 ±0.00	0.80 ±0.40	0.88
zup kiki dax	0.90 ±0.30	0.90 ±0.30	1.00 ±0.00	0.80 ±0.40	0.86
wif kiki zup	1.00 ±0.00	0.90 ±0.30	1.00 ±0.00	0.80 ±0.40	0.86
dax blicket zup	1.00 ±00	0.90 ±0.30	1.00 ±0.00	0.80 ±0.40	0.88
zup blicket lug	0.90 ±0.30	0.70 ±0.50	1.00 ±0	0.80 ±0.40	0.79
wif kiki zup fep	1.00 ±0	0.50 ±0.50	0.00 ±0.00	0.50 ±0.50	0.85
zup fep kiki lug	1.00 ±0.00	0.05 ±0.22	0.00 ±0.00	0.70 ±0.50	0.85
lug kiki wif blicket zup	0.95 ±0.22	0.40 ±0.50	0.00 ±0.00	0.50 ±0.50	0.65
zup blicket wif kiki dax fep	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.70
zup blicket zup kiki zup fep	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.75

Table 8: Colors dataset exact match breakdown for each individual test example. Human results are taken from (Lake et al., 2019)Fig2.

B Learned Lexicons

Here we provide lexicons for each model and dataset (also see Fig. 1 and Fig. 2 for misisng datasets). For COGS, we show a representative subset of words.

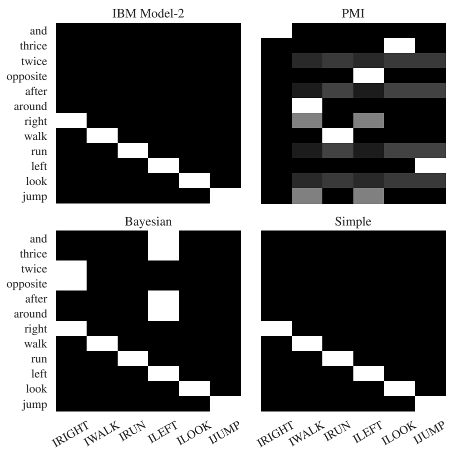

Figure 3: Learned lexicons from SCAN datset *jump* split with $\tau = 0.1$

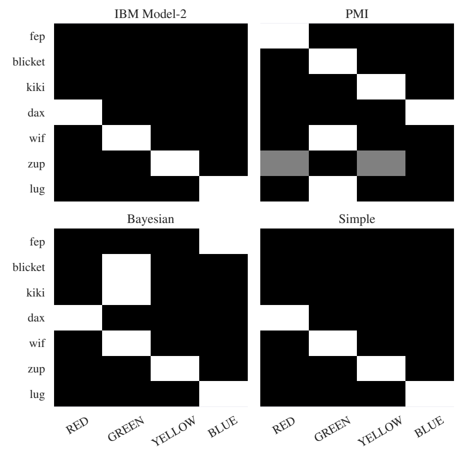

Figure 4: Learned lexicons from Colors datset with $\tau = 0.1$

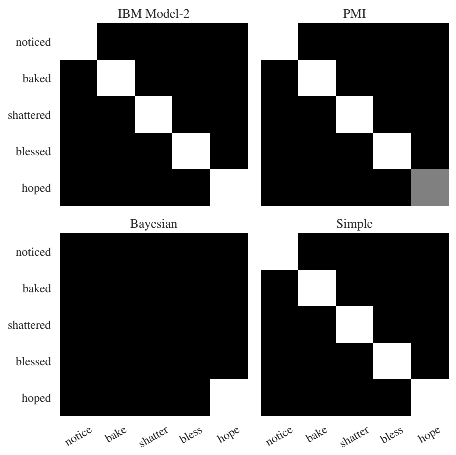

Figure 5: Learned lexicons from COGS datset with $\tau = 0.1$. We only show important rare words resposible for our model's improvements over the baseline.

C Hyper-parameter Settings

C.1 Neural Seq2Seq

Most of the datasets we evaluate do not come with a out-of-distribution validation set, making principled hyperparameter tuning difficult. We were unable to reproduce the results of Kim and Linzen (2020) with the hyperparameter settings reported there with our base LSTM setup, and so adjusted them until training was stabilized (issues with exploding parameters). Like the original paper, we used a unidirectional 2-layer LSTM with 512 hidden units, an embedding size of 512, gradient clipping of 5.0, a Noam learning rate scheduler with 4000 warm-up steps, and a batch size of 512. Unlike the original paper, we found it necessary to reduce learning rate to 1.0, increase dropout value to 0.4, and the reduce maximum step size timeout to 8000.

We use same parameters for all COGS, SCAN, and machine translation experiments. For SCAN, we additionally applied dropout of 0.5 dropout in the last layer of p_{write}.

Since Colors has 14 training examples, we need a different batch size, set to 1/3 of the training set size ($= 5$). Qualitative evaluation of gradients in training time revealed that stricter gradient clipping was also needed ($= 0.5$). Similarly, we decreased warm-up steps to 25 epochs. All other hyper-parameters the same.

C.2 Lexicon Learning

Simple Lexicon Only parameter in simple lexicon is ϵ and it is set to 3 in all experiments.

Bayesian The original work of Frank et al. (2007) did not report hyperparemeter settings or sampler details. We found $\alpha = 2$, $\gamma = 0.95$ and $\kappa = 0.1$ to be effective. M–H proposal distribution inserts or removes a word from the lexicon with 50% probability. For deletions, an entry is removed uniformly at random. For insertions, an entry is added with probability proportional to the empirical joint co-occurrence probability of the input and output tokens. Results were averaged across 5 runs, with a burn-in period of 1000 and a sample drawn every 10 steps.

IBM Model 2 We use the FastAlign implementation provided in (Dyer et al., 2013). We run it with option $-o$ where it optimizes λ that controls diagonality of alignments. We also use intersection to merge forward and reverse runs of same algorithm that is also mentioned in Section 4.1.

D Baseline Results

D.1 GECA

We reported best results for SCAN dataset from reproduced results in (Akyürek et al., 2020). For other datasets (COGS and Colors), we performed a hyperparameter search over augmentation ratios of 0.1 and 0.3 and hidden sizes of {128, 256, 512}. We report the best results for each dataset.

D.2 SyntAtt

We used the public GitHub repository of SyntAtt[5] and reproduced reported results for the SCAN dataset. For other datasets, we also explored ”syntax action” option, in which both contextualized context (syntax) and un-contextualized embeddings (semantics) used in final layer Russin et al. (2019). We additionally performed a search over hidden layer sizes {128,256,512} and depths {1,2}. We report the best results for each dataset.

E Datasets & Evaluation & Tokenization

E.1 Datasets and Sizes

	around_right	jump	COGS	Colors	ENG-CHN
train	15225	14670	24155	14	19222
validation	-	-	3000	-	2402
test	4476	7706	21000	10	2402

E.2 Evaluation

We report exact match accuracies and BLEU scores. In both evaluations we include punctuation. For BLEU we use NLTK [6] library's default implementation.

E.3 Tokenization

We use *Moses* library[7] for English tokenization, and *jieba*[8] library for Chinese tokenization. In other datasets, we use default space tokenization.

F Computing Infrastructure

We use a DGX-2 machine as our computing platform with NVIDIA 32GB VOLTA-V100 GPUs. Average running time among all different experiments is 2.50 hours per run at maximum.

[5](`https://github.com/jlrussin/syntactic_attention`)

[6]`https://www.nltk.org/`

[7]`https://pypi.org/project/mosestokenizer/`

[8]`https://github.com/fxsjy/jieba`