Recursive Routing Networks: Learning to Compose Modules for Language Understanding

Ignacio Cases[1], **Clemens Rosenbaum**[2], **Matthew Riemer**[3], **Atticus Geiger**[1], **Tim Klinger**[3], **Alex Tamkin**[1], **Olivia Li**[1], **Sandhini Agarwal**[1], **Joshua D. Greene**[4], **Dan Jurafsky**[1], **Christopher Potts**[1], **Lauri Karttunen**[1]
[1]Stanford University [2]University of Massachusetts Amherst
[3]IBM Research [4]Harvard University
cases@stanford.edu, cgbr@cs.umass.edu

Abstract

We introduce Recursive Routing Networks (RRNs), which are modular, adaptable models that learn effectively in diverse environments. RRNs consist of a set of functions, typically organized into a grid, and a meta-learner decision-making component called the *router*. The model jointly optimizes the parameters of the functions and the meta-learner's policy for routing inputs through those functions. RRNs can be incorporated into existing architectures in a number of ways; we explore adding them to word representation layers, recurrent network hidden layers, and classifier layers. Our evaluation task is natural language inference (NLI). Using the MULTINLI corpus, we show that an RRN's routing decisions reflect the high-level genre structure of that corpus. To show that RRNs can learn to specialize to more fine-grained semantic distinctions, we introduce a new corpus of NLI examples involving implicative predicates, and show that the model components become fine-tuned to the inferential signatures that are characteristic of these predicates.

1 Introduction

Human cognition has an extraordinary ability to modularize, decomposing problems and solving them by re-composing elements from prior solutions, and this ability is nowhere more evident than in language understanding (Partee, 1984; Janssen, 1997). Most machine learning architectures lack this modularity, which limits their ability to generalize and leaves them susceptible to *catastrophic interference* (McCloskey and Cohen, 1989) – forgetting past skills when acquiring new ones.

We propose to address this need for modularity by applying Routing Networks (Rosenbaum et al., 2017) to natural language understanding. Routing Networks are self-organizing networks with two components (Figure 1): a set of function blocks which can be applied to transform the input, and a router which makes decisions about which function block to apply next. Here we introduce Recursive Routing Networks (RRNs), in which there is a single set of composable functions, recursively chosen by the router. RRNs can be applied to different components of modern language understanding architectures with full end-to-end training. The model jointly optimizes the parameters of selected sub-functions and the meta-learner's policy for how to route inputs through those functions. As a result, individual sub-functions specialize to specific inputs, and paths through the grid of sub-functions can similarly be trained to reflect specific concepts and capabilities.

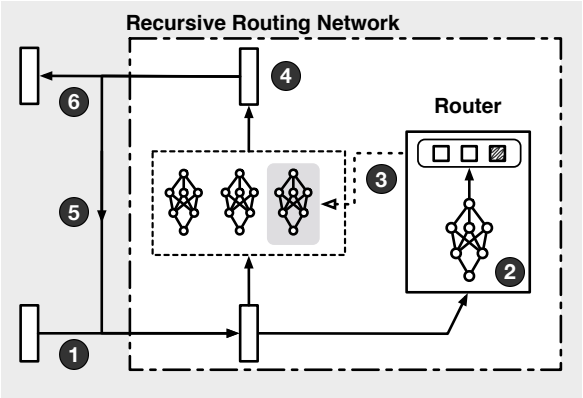


Figure 1: Given a premise–hypothesis pair x, e.g., *t managed to do s* and *t did s* (1), the model needs to learn to predict *entails* (6). The router (2) estimates the value of applying each of the available sub-functions to the input. Given that *manage* has certain semantic properties, the router may select (3) a function f_1 specialized to them. The module is then applied (4), yielding $f_1(x)$. This process repeats (5), now using $f_1(x)$, selecting and applying another sub-function, and so on, until the router is confident in its prediction (6).

RRNs share many intuitions with other modular methods like: Neural Module Networks (Andreas et al., 2015, 2016; Hu et al., 2017), which learn to construct a neural network from pre-defined modules; the Compositional Recursive Learner (Chang et al., 2018), a closely related approach that uti-

Proceedings of NAACL-HLT 2019, pages 3631–3648
Minneapolis, Minnesota, June 2 - June 7, 2019.

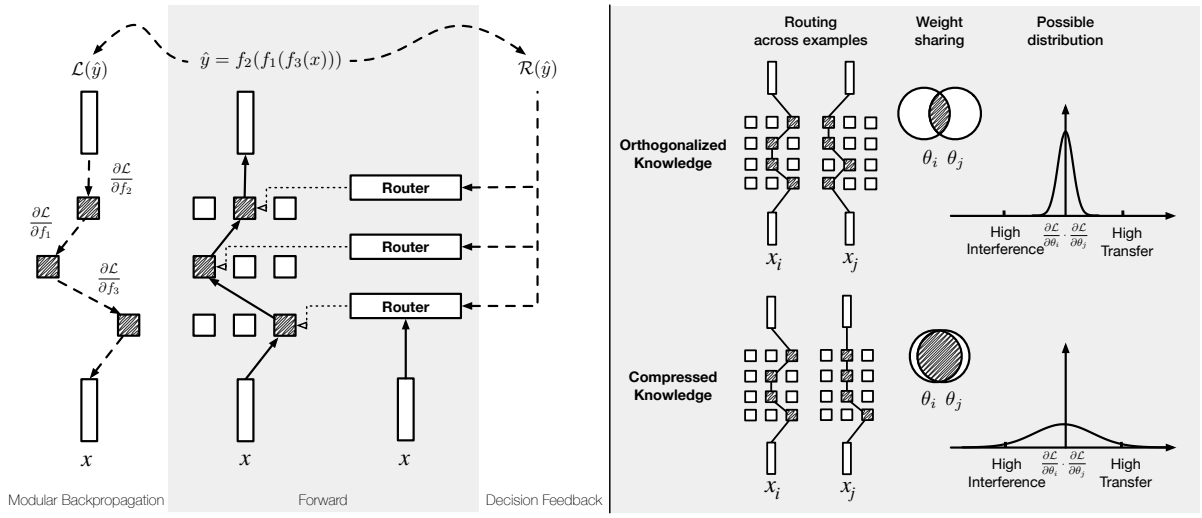


Figure 2: Left: **RRN learning.** Forward (grey background) and backward passes (white background) during the training of a rollout of an RRN with three blocks (width) and limited to depth three (height). In this unrolled version, a *path* is the sequence of function blocks selected by the router. Right: **The dilemma of weight sharing and the transfer–interference trade-off.** When examples are learned using largely separate weights, there is a thin possible distribution of gradient dot products that limits interference, which unfortunately also limits transfer. This is beneficial for unrelated examples, but frustrates the learning of related ones. Conversely, when examples are learned using largely shared weights, there is a wide possible distribution of gradient dot products that allows for both high magnitude transfer and interference. This is beneficial for related examples, but maximizes interference when examples are unrelated. With RRNs, we provide our network with an unprecedented degree of leverage to learn to navigate this trade-off.

lizes curriculum learning to solve arithmetic and vision problems; and ModularNetworks (Kirsch et al., 2018), which extend Routing Networks with an EM-like training approach.

We show how to incorporate RRNs into different neural components (word representation layers, recurrent network hidden layers, classifier layers), and we study their application to natural language inference (NLI), in which premise–hypothesis pairs are labeled for whether the premise entails, contradicts, or is neutral with respect to the hypothesis. We chose this task because reasoning in natural language involves context-sensitive interpretation of words and sentences as well as compositional structure. We make use of the MULTINLI corpus (Williams et al., 2018), which includes text from multiple genres that we expect to condition linguistic senses in complex ways. Our experiments show that RRNs learn policies and components that reflect this genre structure, which leads to superior performance.

We also introduce a new corpus of NLI examples involving implicative constructions like *manage to*, *be able to*, and *fail to* (Karttunen, 1971, 2012). This corpus follows the design of many recent NLI corpora, but with the added challenges of reasoning about implicatives, which have logical *signatures* that interact compositionally with each other and with surrounding semantic operators. Our experiments show that trained RRN model components become fine-tuned to these signatures. Finally, we introduce an extension of the framework which leverages a *Dispatcher* for situations where meta-information is not available at test time, obtaining very promising results.

2 Background and Motivation

RRNs are a natural extension of the Routing Networks introduced by Rosenbaum et al. (2017), themselves part of a larger class of conditional computation models (Bengio et al., 2015). Routing Networks combine trainable modules with a meta-learner called a *router*, which is typically trained using reinforcement learning, though any hard-decision making algorithm could be used. Given an example, the router selects a module and applies it, yielding an activation that can be the input to another routing iteration. As Rosenbaum et al. observe, this can be modelled as a recursive process where the router is applied repeatedly to define complex paths through the modules. The final routed output is passed to additional layers or interpreted as the output of the network (Figure 1).

We are motivated to consider Routing Networks, and in particular recursive variants, in order to address the transfer–interference trade-off

(Riemer et al., 2019). Vanilla neural networks latently attempt to solve a very difficult problem of deciding when to orthogonalize and compress knowledge (Figure 2, right). When two examples are learned with the same weights, there is high potential for transfer as well as interference. This is good for related examples because it maximizes the potential for transfer. However, weight sharing is bad for unrelated examples, as it increases the likelihood of interference. In contrast, when two examples are learned using different weights, there is low potential for transfer and interference. This is beneficial for unrelated examples, but limits the potential for learning about commonalities between related ones. RRNs extend vanilla neural networks by granting them the leverage to navigate this trade-off by making global functional decisions at the module level. As a result, RRNs explicitly make decisions to compress or orthogonalize knowledge between examples by deciding whether to share specific weights.

Thus far, hard-selection routing has not been applied to language domains. Zaremoodi et al. (2018) introduced a *soft* version of routing that falls within a larger class of Mixtures of Experts (MoE) models (Jacobs et al., 1991). However, MoE models differ from RRNs in two crucial ways. First, MoE models generally do not consider the recursive application of functions. The promise is that we can compose functions to reflect the compositional aspects of a problem. Imagine we have a sentence encoding and we want to answer a particular question. We can now condition the router on the question, so that it applies exactly the functions required that translate the encoding to extract the answer. Second, MoE models do not allow for nearly the level of specialization as those based on routing, because they do not eliminate weight sharing across modules and instead only gate the sharing. In practice, this still leads to significant interference across tasks.

Composition of modules has been widely explored for question answering, starting with Andreas et al. (2015). Andreas et al. (2016) learn to assemble a deep neural network on-the-fly from a pre-specified inventory of neural modules using tree-structured layouts based on linguistic analysis. Chang et al. (2018) also explore routing, but with problems described in pseudo-language rather than natural language.

3 Recursive Routing Networks

We now provide a formal presentation of RRNs and show how to incorporate them into existing neural architectures.

3.1 The Routing Paradigm

Formally, the router bases its decision on the tuple $\langle \mathfrak{f}(x), m \rangle$, where x is the sample, $\mathfrak{f}(x) = f_i \circ \cdots \circ f_k(x)$ is the composition of all applied functions from the set of all available functions $\mathcal{F} = \{f_1, \ldots, f_b\}$, and m is a vector that contains meta-information that can be utilized by the router – for example, an embedding for its genre or semantic classification. The router consists of a policy $\pi(\mathcal{F}|\langle \mathfrak{f}(x), m \rangle)$ that determines which of the functions in $\mathcal{F}$ to apply for a given state $\langle \mathfrak{f}(x), m \rangle$. Once a new function f_k is selected, the latest activation (and, thereby, the state) gets updated to $f_k \circ \mathfrak{f}(x)$, and the process repeats.[1]

We focus on the recursive case where the set $\mathcal{F}$ is generally the same for each decision. Apart from being the more general formulation, this form of recursivity allows more weight sharing and better compositional generalization. While there is no necessary upper limit to the number of selections, we found that limiting the number to a maximum of d makes the learning more stable. Consequently, the total number of possible routing paths available is b^d.

Decision Making One of the most important design choices for the router concerns the meta-learning algorithm. Routing is limited to *hard* decision-making algorithms, in particular a stochastic reparameterization using the Gumbel-Softmax function (Jang et al., 2016; Maddison et al., 2016) and reinforcement learning (RL) algorithms. As with other recent approaches to compositional architectures with hard selections (Bengio et al., 2015; Shazeer et al., 2017; Kirsch et al., 2018), routing networks can suffer from *module collapse*, a lack of diversity in the router's decision making. This general problem, common in architectures that jointly train decision making and functions, stems from an early over-estimation of the value of specific functions. This leads to these functions being selected and trained more than others, until these functions are indeed so good

[1] Some architectures have constraints that prevent repeated selection of the same functions in $\mathcal{F}$. In these cases, the router has to be restricted to select from only a compatible subset of $\mathcal{F}$.

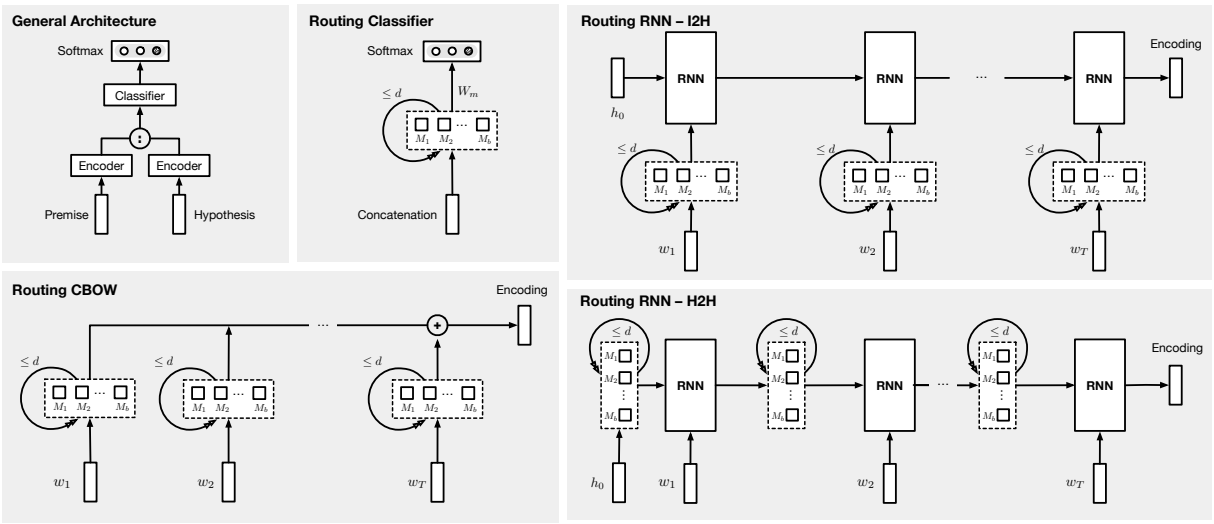


Figure 3: Top left: Our general NLI architecture. Top center: Routing the classifier. Lower left: Word representation routing of a CBOW encoder. Top right: Routing an RNN input transformation. Bottom right: Routing an RNN hidden transformation.

that others are not considered. To mitigate this, we start with a strategy proposed by Rosenbaum et al. (2017): exclusively conditioning the router on meta-information provided by the datasets and storing the router's policy in a table. We found that this stabilizes early training. During later training, we can replace the meta-information hard assignment rule with a meta-information "guesser". We implement this guesser with what Rosenbaum et al. call a *Dispatcher*, an approximation-based sub-policy that estimates the meta-information for each sample and passes it to the corresponding policies (Section 5).

Losses and Training Once the routing procedure terminates, the selected modules are trained using standard optimization techniques, such as backpropagation with stochastic gradient descent (SGD). The decision-making algorithm is either also trained with SGD (Gumbel reparameterization) or trained using reinforcement learning. As we focus on classification problems, our core loss function $\mathcal{L}_{class}(y, \hat{y})$ is the standard cross-entropy loss. We backpropagate this loss along the function path chosen by the router for input x.

Reinforcement Learning Rewards For RL training of the router, we define the reward r for the reinforcement learner to be the sum of the negative classification loss and an additional regularization reward that encourages diversity in the selection of modules:

$$r = -\mathcal{L}_{class}(y, \hat{y}) + r_{reg} \quad (1)$$

For discussion of this reward, see Appendix A.1.

3.2 Architectures

If not otherwise mentioned, we assume that the routed modules are all fully connected layers with the same input and output dimensions. We furthermore assume that the router is always selecting from the same set of functions when defining paths through the routing layers, as this straightforwardly allows recursion in the relevant sense.

Routing Classifiers The simplest method we explore involves routing the layers of a classifier. To do this, we define a fixed set of b fully connected layers (FC), each of the same dimensionality, and we allow the router to choose any path through this set up to a maximum length d (Figure 3, top center). The final activation produced by the chosen path is then densely connected to a non-routed output layer. To further model each example's dependence on its class, we hard-select this output layer based on the meta-information label m for x:

$$W_m \left(\mathrm{FC}_{k_1} \circ \cdots \circ \mathrm{FC}_{k_d}(x)\right) + b_m \quad (2)$$

Routing RNN Encoders In routing RNNs we focus on their two core transformations: from the hidden state at time step t_k to time step t_{k+1}, and from the input to the hidden state. We have designed routing architectures for both of them. In this paper, we start with LSTMs, but the techniques are straightforward to adapt to other cell types.

Figure 3, top right, shows an architecture where we route the input-to-hidden (I2H) transformation. Similarly, Figure 3, bottom right, shows an architecture where we route the hidden-to-hidden

(H2H) transformation. While these transformations are often designed as single fully connected layers, we allow a recursive application of d steps from a selection of b modules to be applied. The selections for the transformations f, i, o, c are tied to be the same. The corresponding transformation (in form of their weight matrices $W_{f/i/o/c}$ for I2H and $U_{f/i/o/c}$ for H2H) becomes:

$$K = \text{FC}_k \circ \cdots \circ \text{FC}_m(x_t), \quad \forall K \in \{W_f, W_i, W_o, W_c, U_f, U_i, U_o, U_c\} \tag{3}$$

Routing CBOW Encoders We also experiment with routing a continuous bag of words (CBOW) encoding. As routing after the main addition is just routing the classifier, we instead add a word-level transformation before the addition (Figure 3, bottom left). This transformation can again be routed recursively (with up to d steps through b modules). The entire CBOW model can be defined as (with $w_1, \ldots, w_t$ as premise or hypothesis):

$$\text{CBOW}_R(w_1, \ldots, w_t) = \sum_{i=1}^{t} \text{FC}_{k_1} \circ \cdots \circ \text{FC}_{k_d}(w_i) \tag{4}$$

Routing Transformers The Transformers model (Vaswani et al., 2017) was recently pretrained as a language model (Radford et al., 2018) achieving impressive results on several NLI datasets. Since the corpus for pretraining is not available, we instead use the parameter-files distributed by the authors.[2] The encoding consists of twelve Transformer-blocks. Each block consists of a convolutional attention layer followed by two convolutional layers (along with several dropout and layer-norm layers). This allows routing at different levels of granularity, of which we investigate two: routing entire blocks and routing the attention step within each block. As we use pre-defined parameters, we cannot apply the blocks or the attention layers recursively. Furthermore, we have to add routing in the fine-tuning phase of using Transformers by creating b copies of each routed module (the depth is necessarily $d = 12$). When fine-tuning, the router then diversifies the initially identical modules.

4 Experiments

We now evaluate the routing architectures introduced in the previous section, along with minimally different non-routing baselines. Our experiments focus on NLI, using the MULTINLI corpus (Williams et al., 2018) and a new English Corpus of Implicatives, the *Stanford Corpus of Implicatives* (SCI). In both, each example is a premise/hypothesis pair labeled with one of *entails*, *contradicts*, or *permits*. What is special about MULTINLI and SCI in the current context is that the examples also have meta-information that can be used to guide the router: a genre label for MULTINLI and an implicative signature for SCI. We expect our routing models to leverage this information during policy and parameter learning.

For all models (except Transformers), we adopt the architecture in Figure 3, top left, in which the premise and hypothesis are processed separately, and the final representations of each are concatenated and fed into the classifier layers. We explore two methods for input processing: (i) pretrained word representations (GloVe; Pennington et al. 2014) and (ii) pretrained contextualized representations (ELMo; Peters et al. 2018). For all non-routed models, we provide explicit access to the genre label via special keywords put in front of each sentence before encoding. The routed models use this label as meta-information.

Unless otherwise specified, we used embedding and hidden dimensions of 300. The classifier consists of three fully connected layers with input and output dimensions of 600 (also when routed). The final layer projects from 600 to the output dimension, 3. The classifier nonlinearities are ReLUs. We train the modules using Adam (Kingma and Ba, 2014) with a learning rate of $1e{-}3$ and the router using SGD with a learning rate of $3e{-}4$ (for additional details, see Appendix A.5). For Transformers, all hyperparameters are determined by the published parameter files.

In experiments with different decision making algorithms (see Appendix A.5), we found that the Gumbel-Softmax reparameterization performed 10% worse on average, with *much* higher variance. QLearning was consistently as good as other more complex RL algorithms, so we report only our QLearning experiments. As Routing Networks have more parameters than their non-routed counterparts, we ran experiments with larger non-routed networks. We found that this did not affect performance, only resulting in more overfitting.

[2] https://github.com/openai/finetune-transformer-lm

Encoding	Embedding	Routing	MULTINLI match	SCI joint	SCI disjoint	SCI mismatch	SCI nested*
CBOW	Glove	None	61.94±0.13	57.26±0.18	55.68±0.47	53.41±0.86	50.98±0.92
	Elmo	None	59.12±0.59	58.28±0.67	57.99±0.90	56.30±1.28	
	Glove	Classifier	51.99±0.17	62.33±0.80	61.17±0.28	51.55±1.61	
	Elmo	Classifier	55.36±0.35	64.29±0.99	61.60±0.61	55.50±0.87	
	Glove	WP	64.38±0.24	*74.95*±0.64	*73.91*±0.54	68.69±0.93	
	Glove	WP+D	**65.84**±0.12	**75.56**±0.77	**74.87**±0.49	**71.08**±0.52	**75.43**±0.29
	Elmo	WP	61.28±0.25	65.91±4.19	61.67±1.30	57.34±0.52	
RNN	Glove	None	**66.53**±0.21	67.04±2.36	63.52±2.00	*59.32*±2.87	57.69±2.75
	Elmo	None	66.13±0.19	65.76±2.16	61.89±2.41	*59.02*±1.92	
	Glove	Classifier	65.60±0.33	**71.02**±0.85	**67.82**±0.81	**60.58**±1.35	
	Elmo	Classifier	59.49±0.55	*68.98*±2.99	*67.23*±2.40	52.96±1.82	
	Glove	I2H	47.79±0.47	53.25±0.28	56.57±1.83	53.99±0.55	
	Elmo	I2H	49.12±1.01	56.13±1.04	56.23±0.17	53.36±0.26	
	Glove	H2H	62.34±0.25	54.89±0.93	53.08±0.57		
	Elmo	H2H	64.04±0.19	59.34±4.41	56.57±1.83	57.61±0.55	
Transformers (pretrained)		None	76.12	88.12	88.07	85.64	
		Classifier	76.50	86.05	86.68	73.45	
		Attention	**76.63**	87.91	**88.45**	**85.95**	
		Block	75.87	**88.22**	87.88	85.33	

Table 1: Results for MULTINLI and SCI with different baselines and their routed versions. We report average accuracy with confidence intervals over five runs with different seeds. For Transformers, we found that finetuning was highly volatile. We therefore report test results from the best-of-5 train models. *All results for nested SCI were computed by fine-tuning the same network previously trained on joint. 'WP' stands for Word Projection, '+D' for Dispatching, 'I2H' for Input-to-Hidden routing, and 'H2H' for Hidden-to-Hidden routing. Italics mark scores whose confidence intervals overlap with the best scores.

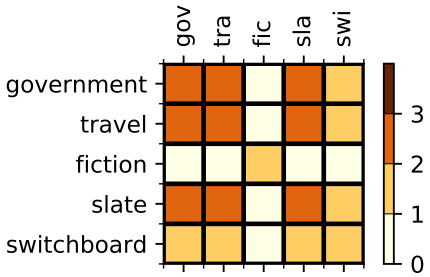


Figure 4: Path (module selection) overlap for MULTINLI between genres with the CBOW GloVe WP model. The diagonal represents the number of function blocks applied for a genre. A maximum of three means that two genres would be routed through the exact same functions.

4.1 Experiments with MULTINLI

The MULTINLI corpus contains 392,702 training examples, 10K dev examples, and 10K test examples. The examples come from 5 genres: fiction, government reports, the *Slate* website, the Switchboard corpus (Godfrey and Holliman, 1997), and Berlitz travel guides. We treat these genre labels as meta-information for the model (Section 3.1).

Our MULTINLI results are given in Table 1, and the learning dynamics in Figure 5. The best model combines the Transformer base model with routing in the attentional layers. Our methods for routing the RNN seem to be less successful, but word-representation routing offers clear benefits with the CBOW base model. Interestingly, the baseline (non-routed) models perform at the same level as the very similar models without genre labels evaluated by Williams et al. (2018). It seems that these models are not able to take advantage of the meta-information. In contrast, RRNs seem to provide the space needed to condition linguistic senses on these labels.

Our hypothesis is that routing will not only lead to better performance on diverse tasks like MULTINLI, but also that the paths – i.e., the sequence of functions selected by the router – followed by the network will reflect high-level task structure. Figure 4 suggests that this is the case for MULTINLI. Here we show the degree of path-overlap for all pairs of genres. As we might expect, government (the 9/11 report), Slate (current affairs), and travel cluster together, as distinct from the two more isolated genres (Switchboard; spoken language) and fiction (mostly from the 20th century).

4.2 Experiments with Implicatives

Karttunen (1971) discovered that implicative constructions, such as *manage* and *waste chance* (e.g. *They wasted their chance to win*), have *signatures*, which characterize the inferences they support in positive and negative contexts. This makes them

the order compelled him to appear as a witness	**entails**	he appeared as a witness
we have missed an opportunity to examine the art market today	**contradicts**	we have examined the art market today
Mr Odinga had not been forced to change his plans	**permits**	Mr Odinga had changed his plans

Table 2: Examples from SCI randomly chosen from the validation set. Each row contains a triplet formed by a premise (left column), a hypothesis (right column), and a label specifying one of the three possible relations (*entails*, *contradicts*, *permits*) holding between premise and hypothesis. The last row contains an example of a probabilistic implicative (see the main text).

particularly informative for NLI predictions.

For instance, the positive sentence *Joan managed to solve the problem* entails *Joan solved the problem*, and the negative sentence *Joan didn't manage to solve the problem* contradicts *Joan solved the problem*, so we say that the verb *manage* has the signature + | − (MacCartney and Manning, 2009; Karttunen, 2016). In contrast, *waste chance* has the opposite signature, since *they wasted the chance to befriend him* contradicts *they befriended him*, and *they didn't waste the chance to befriend him* entails *they befriended him*. There are seven implicative signatures: six were previously known (Karttunen, 1971, 2012), and we found an additional one (+ | +; e.g. *take no time to*). See Appendix C for additional details.

Signatures are compositional: when two or more implicative constructions are composed in a sentence, they create a *nested implicative construction* whose signature is determined by the signatures of the individual verbs (Nairn et al., 2006). For example, *John managed to remember to get the keys* entails *John got the keys*, where the nested implicative *manage to remember* has the overall signature + | −. We also see a more limited form of compositionality inside phrasal implicatures; their signatures are often largely determined by the lexical semantic family of their constituent words. Therefore, signatures make implicatives ideal for evaluating different degrees of compositional generalization with RRNs, as they provide valuable meta-information (Section 3.1).

Statistic	SCI
Validated pairs	41.15%
Unanimous gold label	94.93%
Gold matches author	98.61%
Gold does not match author	1.39%
No gold label	0
Fleiss κ coefficient	0.95
entails	0.32
contradicts	0.30
permits	0.38

Table 3: SCI statistics. Top: Percentage of validated pairs and basic agreement. Bottom: Fleiss κ coefficients and proportion of all assignments made to the corresponding label.

4.2.1 A Corpus of Implicative Constructions

Our SCI dataset contains ≈10K premise–hypothesis pairs. All seven signature types are represented, in addition to pragmatic signatures, which have inferential biases (see Appendix C).

We provide SCI[3] in three versions for all single and phrasal implicatives.[4] In the *joint* version, the underlying distribution of implicatives is shared across train, validation, and test splits. In the *disjoint* version, a different subset of implicatives is used in train from those used in validation and test. This allows us to test generalization to unseen constructions. Although disjoint with respect to constructions, the constructions are carefully distributed so that all the underlying signatures and most lexical items are represented in all splits. The lexical items that make up implicative constructions overlap between the splits. For example, *take vow* appears only in training and validation, while *make vow* only appears in test. The last version is *mismatch*, where different subsets of the signatures are present in training/validation and test (see appendix C for more details).

Data collection for SCI proceeded as follows: we collected at least 12, and most often 20, seed premises from examples found in Google Books and on the web. For each example, six hypotheses were created by expert annotators.[5] These six examples were constructed in two steps: first, the premise was taken to be the seed sentence, and three hypotheses were created to exhaust the label space in relation to the premise. Second, the premise was taken to be the *negation* of the seed sentence, and a different set of three hypotheses was created to also exhaust the label space with respect to this negated sentence. For example, one of the seeds for *manage* was *I managed to see who it was*. From this, annotators produced by hand a

[3] https://nlp.stanford.edu/projects/sci/

[4] We provide nested constructions as a separate extension, with the exception of a few nested implicatives that were used in the development version of this corpus, for which the results are presented here. In the final version of the corpus, single and phrasal implicatives will be separated from nested implicatives. See the Appendix for more details.

[5] Native speakers of English trained in semantics at the undergraduate level, with expertise in implicatives.

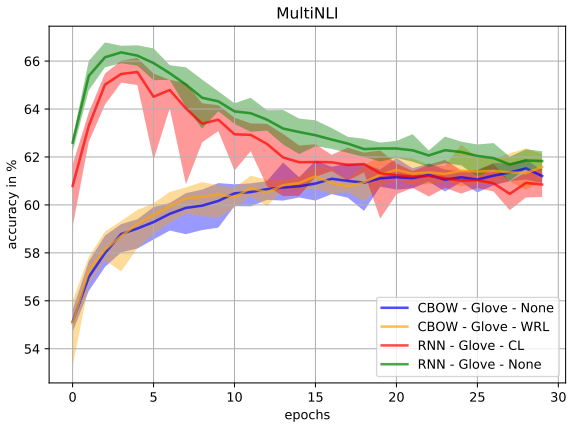


Figure 5: Learning on MULTINLI with GloVe inputs.

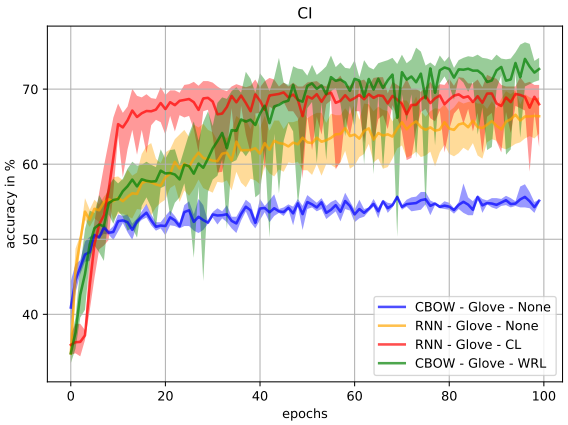


Figure 7: Learning on SCI (joint) with GloVe inputs.

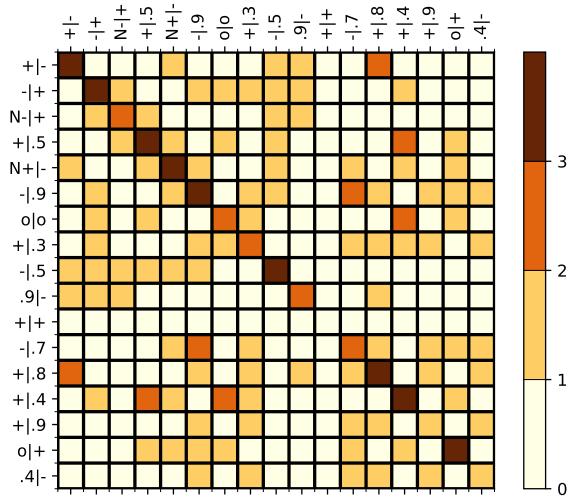


Figure 6: The path-overlap between different signatures on SCI, using the CBOW GloVe WP model, for $b = 4$, $d = 3$. N+|− and N−|+ are nested signatures.

negated seed, *I did not manage to see who it was*. Then three hypotheses were generated for each of these two premises. Finally, each example was labeled by the author of the hypothesis examples.

Validation was performed on a randomly selected subset of constructions. Two additional votes were cast by different linguists, resulting in three label votes for each example in the subset. The gold label was then defined to be the majority class from this set of votes. The Fleiss κ shows high inter-annotator agreement (Table 3).

4.2.2 Implicatives Results and Analysis

For each example in SCI, we use its associated implicative signature as the meta-information label. This serves as a subtler kind of semantic information than genre. The learning dynamics of different models are shown in Figure 7. As Table 1 shows, RRNs lead to considerable gains in performance over most benchmarks. Recursive routing of the classifier helps consistently with all models. Even simple word-level routing yields major improvements for CBOW. As with MULTINLI, we believe that the modules allow the conditioning of linguistic senses. Routing at the word-level for sequential models seems suboptimal here. The training accuracies ($> 99.5\%$) suggest that the problem is overfitting. Intuitively, routing can assign too many exclusive parameters to each class; and as the samples within these classes are similar to begin with, remembering them can be easier for the network than for non-routed networks.

We expect examples with similar signatures to be routed along similar paths. Figure 6 summarizes the path-overlap between different signatures. Some similar paths involve reversals of polarity, which calls for further scrutinity. However, many of these similarities make intuitive sense. For example, `+|.4` and `+|.5` are highly similar, as are `−|.7` and `−|.9`. In addition, the isolation of the unusual signature `+|+` (limited to just two constructions) seems expected, as does the affinity of the nested signatures `N+|−` and `N−|+` to their unnested counterparts. However, as shown in Table A3, some signatures only contain very few samples, resulting in highly noisy routing paths.

4.3 Navigating the Transfer–Interference Trade-off

We now seek to provide intuitive explanation for how RRNs help in navigating the transfer–interference trade-off. We have characterized the trade-off in terms of controlling weight sharing between examples. While baseline networks can learn to navigate this trade-off if they are optimized for the appropriate objective (Riemer et al.,

2019), in general these models do not have available supervision on how to do this and so optimize greedily for the current example. As such, early experiments with the baseline, non-routed models show that examples with neutral signatures, such as ○ | ○ and ○ | +, are the first to be learned. We noticed that baseline sequence models learned to classify *take vow* much earlier in the training process, as the word *vow* is lexicalized through training on *make vow* examples. However, its performance decreases over time as *take* is lexicalized through training on examples of other implicatives with different signatures, such as *take chance*. This behavior is the characteristic outcome of catastrophic interference: learning *take chance* results in decreased performance on *take vow* as the two examples interfere. This is a particularly revealing instance as these two phrasal verbs are similar on the surface, but have quite different semantic properties, as reflected by their signatures. However, given that RRNs are able to route them differently (Figure 6), interference is less likely. It is also possible that routing helps with the transfer related to similar examples, an avenue we want to explore in the future.

5 Prediction without Meta-Information

We have shown that high-quality meta-information can be extremely useful. Unfortunately, it is often not available at test-time. To compensate, we evaluated an RRN extension in which an additional neural network module is trained to assign examples to meta-information classes (genres for MULTINLI; signatures for SCI). When we trained this model jointly, we found that it was unstable and did not perform well. However, if this model is introduced after RRN training on examples with known meta-information, then the results are extremely promising. We call this *Dispatcher Training* ('+D'), borrowing similar terminology from Rosenbaum et al. (2017). Table 1 includes an initial evaluation of this variant with a CBOW base model and WP routing. As we can see, accuracy actually *increases* by a small amount over WP alone.

Additionally, having +D allows the network to generalize better to unseen examples and unseen patterns. Consider the relative performance drop for CBOW WP+D from the full joint SCI dataset (75.56%) to disjoint (−0.69%) and from disjoint (74.87%) to mismatch (−3.79%), and compare this to the plain WP version: full (74.95%) to disjoint (−1.04%), and disjoint (73.91%) to mismatch (−5.22%)

6 Conclusion

This paper introduced Recursive Routing Networks and showed how to incorporate them into a variety of different neural architectures; we explored a range of possibilities for this, and the techniques generalize to other options straightforwardly. Our evaluations focused on NLI. We showed in particular that our RRNs can effectively leverage the meta-information in the MULTINLI corpus and in our new corpus focused on implicatives; not only do RRNs use this information to achieve superior accuracy, but they also learn sub-structure that reflects this high-level information, and our Dispatcher variant extends the framework to situations where the relevant meta-information is not available for testing. We believe exploring more powerful variants of dispatching is an interesting avenue for future work, as is pretraining routing models on language model tasks using large corpora. It is our hope that these lessons extend to other richly compositional, context-sensitive language understanding tasks.

Acknowledgements

We thank George Supaniratisai, Arun Chaganty, Kenny Xu and Abi See for valuable discussions, and the anonymous reviewers for their useful suggestions. Clemens Rosenbaum was a recipient of an IBM PhD Fellowship while working on this publication. We acknowledge the Office of the Vice Provost for Undergraduate Education at Stanford for the summer internships for Atticus Geiger, Olivia Li and Sandhini Agarwal. This research is based in part upon work supported by the Stanford Data Science Initiative, by the NSF under Grant No. BCS-1456077, by the NSF Award IIS-1514268, and by the Air Force Research Laboratory and DARPA under agreement number FA8750-18-2-0126. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory and DARPA or the U.S. Government.

References

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2015. Deep compositional question answering with neural module networks. *CoRR*, abs/1511.02799.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Learning to compose neural networks for question answering. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1545–1554. Association for Computational Linguistics.

Rebekah Baglini and Itamar Francez. 2016. The implications of managing. *Journal of Semantics*, 33:541–560.

Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. 2015. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*.

Michael B Chang, Abhishek Gupta, Sergey Levine, and Thomas L Griffiths. 2018. Automatically composing representation transformations as a means for generalization. *arXiv preprint arXiv:1807.04640*.

Michael L. Geis and Arnold M. Zwicky. 1971. On invited inferences. *Linguistic Inquiry*, 2(4):561–566.

John J. Godfrey and Ed Holliman. 1997. Switchboard-1 release 2. Linguistic Data Consortium, Catalog #LDC97S62.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. 2017. Learning to reason: End-to-end module networks for visual question answering. *CoRR*, abs/1704.05526.

Robert A. Jacobs, Michael I. Jordan, and Andrew G. Barto. 1991. Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks. *Cognitive Science*, 15(2):219–250.

Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with Gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Theo M. V. Janssen. 1997. Compositionality. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*, pages 417–473. MIT Press and North-Holland, Cambridge, MA and Amsterdam.

Lauri Karttunen. 1971. Implicative verbs. *Language*, 47(2):340–358.

Lauri Karttunen. 2012. Simple and phrasal implicatives. In **SEM 2012*, pages 124–131. Association for Computational Linguistics, Montréal, Canada.

Lauri Karttunen. 2016. Presupposition: What went wrong? In Mary Moroney, Carol-Rose Little, Jacob Collard, and Dan Burgdorf, editors, *Semantics and Linguistic Theory (SALT) 26*, pages 705–731. Cornell University, Ithaca, NY.

Lauri Karttunen and Stanley Peters. 1979. Conventional implicature. In Choon-Kyu Oh and David A. Dinneen, editors, *Syntax and Semantics, Volume 11: Presupposition*, pages 1–56. Academic Press, New York.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Louis Kirsch, Julius Kunze, and David Barber. 2018. Modular networks: Learning to decompose neural computation. *arXiv preprint arXiv:1811.05249*.

Bill MacCartney and Christopher D. Manning. 2009. An extended model of natural logic. In *The 8th International Conference on Computational Semantics (IWCS-8)*, pages 140–156. University of Tilburg, Tilburg, Netherlands.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. 2016. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*.

Michael McCloskey and Neal J. Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation*, 24:109 – 165.

Rowan Nairn, Lauri Karttunen, and Cleo Condoravdi. 2006. Computing relative polarity for textual inference. In Johan Bos and Alexander Koller, editors, *Inference in Computational Semantics (ICoS-5)*, pages 67–76. University of Manchester, Manchester, UK.

Barbara H. Partee. 1984. Compositionality. In Fred Landman and Frank Veltman, editors, *Varieties of Formal Semantics*, pages 281–311. Foris, Dordrecht.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237. Association for Computational Linguistics.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. OpenAI.

Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. 2019. Continual learning by maximizing transfer and minimizing interference. *ICLR*.

Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. 2017. Routing networks: Adaptive selection of non-linear functions for multi-task learning. *arXiv preprint arXiv:1711.01239*.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

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

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics.

Poorya Zaremoodi, Wray Buntine, and Gholamreza Haffari. 2018. Adaptive knowledge sharing in multi-task learning: Improving low-resource neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 656–661.

Appendices

A Architecture Details

A.1 The reward design

Rosenbaum et al. (2017) found that it was helpful in incentivizing different routing strategies to add a regularization reward to each action made. This reward is constructed by computing the average count of choosing a particular module, multiplied by a scaling factor. We also use a similar reward. In particular, we compute it for module m_k and action a as follows:

$$r_c(m_k) = \begin{cases} (1-\alpha) * r'_c(m_k) + \alpha & \text{if } k = a \\ r'_c(m), & \text{otherwise} \end{cases}$$

As we want this reward to sum to 1 over all modules, we normalize. Additionally, we want to scale this reward to stay constant even for larger routing depths. The result is therefore scaled as such:

$$r_c = \frac{r_c}{d \sum r_c}$$

A.2 Losses

To make the backpropagation step to train the network more efficient, we combine the losses

$$\mathcal{L}_{total}(x, \hat{y}) = \mathcal{L}_{class}(x, \hat{y}) + \mathcal{L}_{RL}(x, r) \quad (5)$$

where $\mathcal{L}_{RL}(x, r)$ is the loss defined by the chosen RL algorithm.

A.3 Routing RNNs

We modify LSTMs as originally defined by Hochreiter and Schmidhuber (1997):

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \quad (6)$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \quad (7)$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \quad (8)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \quad (9)$$

$$h_t = o_t \circ \sigma_h(c_t) \quad (10)$$

However, analogous architectures can be implemented for GRUs or different LSTMs. Here, either architecture has two core transformations – one from the hidden state at time step t_k to time step t_{k+1}, and one from the input (in our case, word embeddings) to the hidden state. We have designed architectures that route either.

The resulting equations (6), (7), and (8) for the input-to-hidden routing model illustrated in Figure 3, top right, become (with g being a placeholder for f, i, o):

$$g_t = \sigma_g(\text{FC}_{g,k_d} \circ \text{FC}_{g,k_{d-1}} \circ \cdots \text{FC}_{g,k_d} x_t + U_g h_{t-1} + b_g) \quad (11)$$

Similarly, the equations (6), (7), and (8) for the hidden-to-hidden routing model in Figure 3, bottom right, become (with g being a placeholder for f, i, o):

$$g_t = \sigma_g(W_g x_t + \text{FC}_{g,k_d} \circ \text{FC}_{g,k_{d-1}} \circ \cdots \text{FC}_{g,k_d} h_{t-1} + b_g) \quad (12)$$

A.4 Routing Transformers

The following describes the two different approaches for routing transformers. When routing entire transformer blocks (TB), we have:

$$\mathcal{F}_{k_t} = \{\text{TB}_{t,1}, \text{TB}_{t,2}, \ldots, \text{TB}_{t,b}\} \quad (13)$$

However, as the routing functions for routing the attention layers are interleaved with the other block-layers (BL), the basic routing chain changes to:

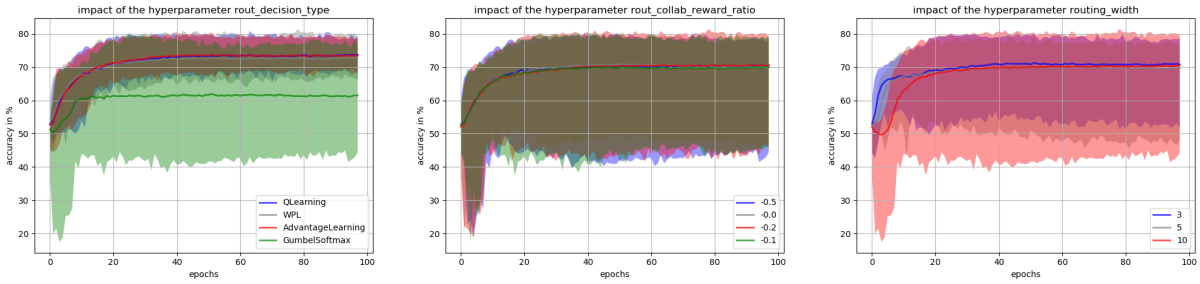
$$f_R(x) = f_{k_1} \circ \text{BL} \circ \ldots \circ \text{BL} \circ f_{k_d}(x) \circ \text{BL} \quad (14)$$

and the routing module sets are (with AL as attention layers):

$$\mathcal{F}_{k_t} = \{\text{AL}_{t,1}, \text{AL}_{t,2}, \ldots, \text{AL}_{t,b}\} \quad (15)$$

A.5 Hyperparameter Search

We found that training the router with Adam can become unstable and stay at random baseline performance. In contrast, we found that using Adam for the modules increased performance by around 1%. When training a routing utilizing the Gumbel-Softmax reparameterization, we tested Adam and SGD exclusively, and settled for Adam. We found the module learning rate of $1e{-}3$ and the router learning rate of $3e{-}4$ by performing sweeps over the learning rate from $1e{-}2$ to $1e{-}4$ for the module training and from $1e{-}2$ to $1e{-}5$ for the router training, roughly by decrements of factor 3. In general, we found a branching size of $b = 3$ to work best (although performance for $b = 5$ was nearly identical), after sweeps for $b \in \{2, 3, 5, 10, 15, 20\}$. As we also specify the maximum recursion depth, we generally allow up to $d = 3$ recursive steps. Higher ds add to training time, but neither increase nor decrease performance significantly, as the router (nearly) never learns to apply more than three transformations anyway. Figure A1 shows the effects of the value of the decision making algorithm, the value of the regularization reward, and of the routing width. Most notably, Gumbel-Softmax routing performs



(a) Effect of the decision-making algorithm.

(b) Effect of the regularization reward value.

(c) Effect of the routing width.

Figure A1: Encoder Routing Strategies.

on average 10% worse than RL algorithms, with variance that is nearly an order of magnitude larger. Also, larger routing width has similar expected performance but increases the variance. The value of the regularization reward has little effect on expected performance.

B Implicatives Experiments

Routed models that achieve good performance on SCI show a strong indication of proper learning of entailment and contradiction, and some difficulties with neutral cases, as indicated by the confusion matrix in Table A1. An inspection of common errors of the CBOW WP model shows a number of cases related to the probabilistic component of the signature. For example, *Jerry got a chance to pitch* was marked as contradicting *Jerry did not manage to pitch*, when the gold label is indeed *permits*. In this case, if the premise is true then there is a small chance that the author would regard the hypothesis as false and therefore predict *contradicts* as the model did. Other errors involve coreference, such as the prediction of entailment between *you took the time to review my presentation* and *you reviewed his presentation*.

Relation	entailment	contradiction	neutral	Totals
entailment	35	2	10	47
contradiction	2	33	5	40
neutral	7	6	28	41
Totals:	44	41	43	128

Table A1: Confusion Matrix.

C Implicatives Corpus

Implicative constructions yield a judgment about the truth of their complement clause under at least one polarity. Some are simple verbs like *manage* and *fail*, while some are phrasal constructions like *meet one's duty* and *waste a chance*.

C.1 Signatures

The signature of an implicative indicates the relation between the matrix clause and complement clause. The symbol + indicates an entailment of the complement clause, the symbol − indicates an entailment of the negation of the complement clause, and the symbol *o* indicates no entailment relation to the complement clause. A signature consists of a left symbol, which indicates the entailment relation to the complement clause when the matrix clause is in a positive environment, and the right symbol, which indicates the entailment relation to the complement clause when the matrix clause is in a negative environment.

For example, in the notation of MacCartney and Manning (2009) and Karttunen (2016), the signature of *manage* is + | −. This means a sentence with *manage* entails its complement in positive contexts, and contradicts its complement in negative contexts. As such, *Joan didn't manage to solve the problem* contradicts *Joan solved the problem*.

Fail and *lack foresight* have the implicative signature − | +. *NP failed/lacked the foresight to VP* contradicts *NP VPed* and entails *NP didn't VP*. This is negative entailment under positive polarity (−), positive entailment under negative polarity (+).

Two-way implicatives like *manage* and *fail* yield an entailment under positive and negative polarity. One-way implicatives such as *force* (+ | ○), *prevent* (− | ○), *be able* (○ | −), and *hesitate* (○ | +) yield an entailment, + or −, only under one polarity and no entailment (○) under the other.[6]

[6]In addition to the entailment properties that we focus on in this study, most implicative constructions also have another component of meaning, sometimes called *presupposition* (Karttunen, 1971; Baglini and Francez, 2016), some-

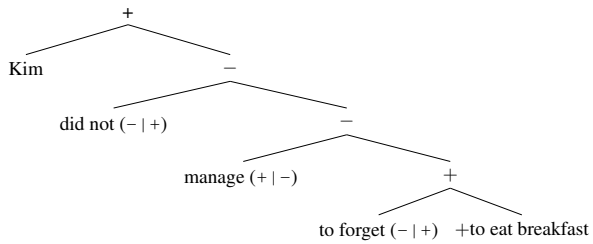


Figure A2: Computation of the inference *Kim ate breakfast* entailed by the premise *Kim did not manage to forget to eat breakfast*.

There are seven known implicative signatures, six of them documented earlier (Karttunen, 1971, 2012). In the course of collecting data for this study, we came across a new implicative construction that had been missed in all previous studies. We code *take no time* and *waste no time* as + | + because, in the vast majority of cases we looked at, the presence or absence of negation makes no difference. *I wasted no time to jump on board* and *I didn't waste no time to jump on board* both mean that I jumped on board quickly.

C.2 Pragmatic Implicatives

In cases where there is no logical entailment, one-way implicatives may give rise to an *invited inference* (Geis and Zwicky, 1971). *I was able to speak but just didn't know what to say* is not a contradiction. But if the speaker leaves out the *but*-clause, she probably means that not only was she able to speak, but she in fact did speak. There is no invited inference in cases like *Bill Clinton had a chance to kill Bin Laden before the September 11 attacks*, because it is common knowledge that the attempt to do so failed. Insofar as these cases depend on an encyclopedic knowledge about the world, as opposed to a dictionary/lexical knowledge, we use the terminology *pragmatic implicatives* for this type of verbs.

The likelihood of invited inferences varies greatly depending on the construction. They are very likely with *be able*, less likely with *have chance*, and very unlikely with *hesitate*. To better match actual usage, we assign to *be able* not the semantically correct ○ | − signature but .9 | −, reflecting our estimation that there is a .9 probability that the author is using the construction as if it were a two-way + | − implicative. We code *have chance* as .4 | − and *hesitate* as ○ | +.

C.3 Nested Implicatives

Implicative constructions can be nested inside of other implicative constructions, as in *Stan managed to fail to propose to Carol*. The meaning of nested implicatives is compositional; the signature of a nested implicative can be determined algorithmically by the signatures of the single implicatives that compose it (Nairn et al. 2006; see Figure A2). Since *manage* has the signature + | − and *fail* is − | +, the nested constructions *manage to fail* and *fail to manage* are both − | +. Thus, the example above entails that Stan did not propose to Carol, as does *Stan failed to manage to propose to Carol*. The entailments are the same, but the implications associated with *manage* and *fail* differ depending on the nesting.

Some implicative constructions – e.g., *manage* and *fail*, *remember*, and *forget* – can be nested in one another. In many cases, only one order makes sense. It is easy to find examples of *happen to turn out* – e.g., *it happened to turn out to be a success* – but a search for *turn out to happen* sentences does not turn up any. One can even find triply nested implicative constructions, such as *don't forget to remember not to forget to watch the finale of series 2 tonight*, but they are rare.

The single-word, phrasal implicatives, and (a few) nested implicatives used in our data set are listed in Table A3. The release version of SCI will contain additional doubly-nested constructions (the complete list is in Table A4), and will have additional splits.

C.4 Corpus Versions and Extension

The three different versions of the dataset described in the article are *joint*, *disjoint*, and *mismatch*. The *disjoint* test set contains the implicatives listed in Table A2, and thus the implicatives listed there are not contained in train and validation.

The *mismatch* version of the SCI corpus contains all signatures in validation and test, but only a restricted set for train. Table A5 contains the subsets available on each split.

times *conventional implicature* (Karttunen and Peters, 1979). For example, *Marco failed to show up for work* implicates that Marco tried or was expected to show up for work. The *NatLog* system (MacCartney and Manning, 2009) computes entailments of implicatives but no presuppositions. No current NLP system does. The difference between entailment and presupposition/conventional implicature can be made with the question *Did Marco fail to show up for work?* This question asks whether Marco showed up for work or not; *fail* indicates that the speaker thinks that Marco tried or was expected to come to work.

In addition to the versions, we included all the nested constructions as a separate extension to the dataset using the same format. This expansion can then be easily used as a test set for generalization to longer sequences.

Figure A3 show the distribution of lengths, counted in word tokens, for premises and hypotheses. Figure A4 shows the distribution of lengths for both premises and hypotheses.

Construction	Sign	Examples
break promise	− | +	90
exploit opportunity	+ | −	120
fail obligation	− | +	60
fulfill promise	+ | −	90
grab occasion	+ | −	120
make vow	○ | ○	91
manage	+ | −	90
meet duty	+ | −	90
miss occasion	− | +	120
neglect	− | +	91
refrain	− | +	120
require	+ | . 3	120
seize opportunity	+ | −	120
take opportunity	+ | −	91
use occasion	+ | −	120
waste chance	− | +	91
waste no time	+ | +	76
waste opportunity	− | +	91
Constructions: 18	Total	1791

Table A2: Disjoint dataset in test set in SCI.

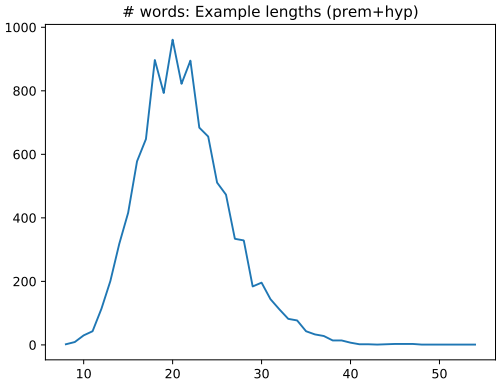


Figure A3: Distribution of sentence lengths in number of words for all verbs in SCI.

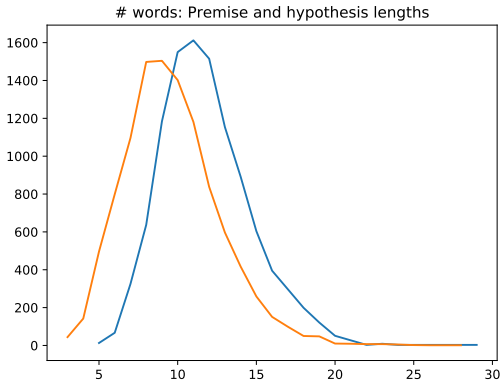


Figure A4: Distribution of lengths for premises (blue) and hypothesis (orange) in SCI.

Construction	Sign	Examples	Construction	Sign	Examples
be forced	+|−	128	keep promise	+|−	90
be kept from	−|.9	132	lack foresight	−|+	91
be made	+|.5	126	lose opportunity	−|+	90
be obstructed from	−|.9	126	make	+|.5	126
be prevented	−|.7	87	make promise	○|○	91
be required	+|.3	120	make vow	○|○	91
be stopped	−|.5	120	manage	+|−	90
bother	+|−	91	manage to miss opportunity	−|+	120
breach contract	−|+	120	meet duty	+|−	90
breach promise	−|+	120	meet obligation	+|−	90
break pledge	−|+	73	meet promise	+|−	88
break promise	−|+	90	miss chance	−|+	90
bring oneself	+|−	120	miss occasion	−|+	120
cause	+|.5	120	miss opportunity	−|+	91
coerce	+|.9	120	neglect	−|+	91
compel	+|.4	120	neglect duty	−|+	120
convince	+|.8	120	neglect occasion	−|+	120
dare	+|−	91	obey order	+|−	91
disobey order	−|+	87	overlook opportunity	−|+	120
exploit occasion	+|−	120	prevent	−|.7	60
exploit opportunity	+|−	120	proceed	+|−	120
fail	−|+	91	refrain	−|+	120
fail obligation	−|+	60	refuse to take advantage	−|+	114
fail to take advantage	−|+	120	remember	+|−	91
follow order	+|−	90	require	+|.3	120
force oneself	+|−	120	seize occasion	+|−	120
forget	−|+	90	seize opportunity	+|−	120
fulfill promise	+|−	90	stop	−|.5	120
get chance	.9|−	90	succeed	+|−	120
get	+|−	119	take advantage of opportunity	+|−	120
grab occasion	+|−	120	take chance	+|−	91
grab opportunity	+|−	120	take no time	+|+	76
happen	+|−	92	take occasion	+|−	120
have chance	.4|−	89	take opportunity	+|−	91
have courage	+|−	91	take time	+|−	88
have	+|.5	120	take vow	○|○	91
have foresight	+|−	90	turn out	+|−	153
have gall	+|−	90	use occasion	+|−	120
have time	.9|−	90	use opportunity	+|−	120
have wherewithal	+|−	90	venture	+|−	120
hazard	+|−	120	waste chance	−|+	91
hesitate	○|+	91	waste money	+|−	91
ignore duty	−|+	120	waste no time	+|+	76
ignore opportunity	+|−	120	waste occasion	−|+	120
jump on occasion	+|−	120	waste opportunity	−|+	91
keep from	−|.9	132	waste time	+|−	91
			Constructions: 92	Total	9671

Table A3: Implicatives in joint SCI.

Construction	Sign	Examples
be able to bring oneself	+ | −	160
fail to have courage	− | +	66
fail to have means	− | +	36
fail to have wherewithal	− | +	18
fail to manage	− | +	90
fail to meet obligation	− | +	54
fail to obey order	− | +	51
fail to take advantage	− | +	120
forget to remember	− | +	48
manage to fail	− | +	54
manage to forget	− | +	18
manage to have gall	+ | −	18
manage to have power	+ | −	18
manage to have strength	+ | −	54
manage to have wherewithal	+ | −	18
manage to miss opportunity	− | +	166
manage to remember	+ | −	53
manage to turn out	+ | −	36
refuse to take advantage	− | +	114
remember to forget	− | +	359
remember to take time	+ | −	54
take advantage of opportunity	+ | −	120
Constructions: 22	Total	1725

Table A4: Nested Implicatives collected to appear in the release version of SCI.

Split	Signatures
train	−|+, +|−, ○|−, ○|○, +|.5, .4|−, −|.7
dev/test	−|+, +|−, ○|−, ○|○, +|.3, .4|−, +|.4, +|.5, −|.5, −|.7, +|.8, +|.9, −|.9, .9|−

Table A5: Mismatch Splits in SCI.