PAIRED EXAMPLES AS INDIRECT SUPERVISION IN LATENT DECISION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Compositional, structured models are appealing because they explicitly decompose problems and provide interpretable intermediate outputs that give confidence that the model is not simply latching onto data artifacts. Learning these models is challenging, however, because end-task supervision only provides a weak indirect signal on what values the latent decisions should take. This often results in the model failing to learn to perform the intermediate tasks correctly. In this work, we introduce a way to leverage *paired examples* that provide stronger cues for learning latent decisions. When two related training examples share internal substructure, we add an additional training objective to encourage consistency between their latent decisions. Such an objective does not require external supervision for the values of the latent output, or even the end task, yet provides an additional training signal to that provided by individual training examples themselves. We apply our method to improve compositional question answering using neural module networks on the DROP dataset. We explore three ways to acquire paired questions in DROP: (a) discovering naturally occurring paired examples within the dataset, (b) constructing paired examples using templates, and (c) generating paired examples using a question generation model. We empirically demonstrate that our proposed approach improves both in- and out-of-distribution generalization and leads to correct latent decision predictions.

1 INTRODUCTION

Developing models that are capable of reasoning about complex real-world problems is challenging. It involves decomposing the problem into sub-tasks, making intermediate decisions, and combing them to make the final prediction. While many approaches develop black-box models to solve such problems, we focus on compositional structured models as they provide a level of explanation for their predictions via interpretable latent decisions, and should, at least in theory, generalize better in compositional reasoning scenarios. For example, to answer *How many field goals were scored in the first half?* against a passage containing a football-game summary, a neural module network (NMN; Andreas et al., 2016) would first ground the set of *field goals* mentioned in the passage, then filter this set to the ones scored *in the first half*, and then return the size of the resulting set as the answer.

Learning such models using just the end-task supervision is difficult, since the decision boundary that the model is trying to learn is complex, and the lack of any supervision for the latent decisions provides only a weak training signal. Moreover, the presence of dataset artifacts (Lai & Hockenmaier, 2014; Gururangan et al., 2018; Min et al., 2019, *among others*), and degeneracy in the model, where incorrect latent decisions can still lead to the correct output, further complicates learning. As a result, models often fail to predict meaningful intermediate outputs and instead end up fitting to dataset quirks, thus hurting generalization (Subramanian et al., 2020).

We propose a method to leverage related training examples to provide an indirect supervision to these intermediate decisions. Our method is based on the intuition that related examples involve similar sub-tasks; hence, we can use an objective on the outputs of these sub-tasks to provide an additional training signal. Concretely, we use *paired examples*—instances that share internal substructure—and apply an additional training objective relating the outputs from the shared substructures resulting from partial model execution. Using this objective does not require supervision for the output of the shared substructure, or even the end-task of the paired example. This additional training objective imposes weak constraints on the intermediate outputs using related examples and provides the model with a richer training signal than what is provided by a single example. For example, *What was the shortest field goal?* shares the substructure of finding all *field goals* with *How many field goals were scored?*. For this *paired example*, our proposed objective would enforce that the output of this latent decision for the two questions is the same.

We demonstrate the benefits of our paired training objective using a textual-NMN (Gupta et al., 2020a) designed to answer complex compositional questions on DROP (Dua et al., 2019), a dataset requiring natural language and symbolic reasoning against a paragraph of text. While there can be many ways of acquiring paired examples, we explore three directions for DROP. First, we show how naturally occurring paired questions can be automatically found from within the dataset. Further, since our method does not require end-task supervision for the paired example, one can also use data augmentation techniques to acquire paired questions without requiring additional annotation. We show how paired questions can be constructed using simple templates, and how a pretrained question generation model can be used to generate paired questions.

We empirically show that this paired training objective leads to overall performance improvement of the NMN model. While each kind of paired data acquisition leads to improved performance, we find that combining paired examples from all techniques leads to the best performance (§5.1). We quantitatively show that using our paired objective results in significant improvement in predicting the correct latent decisions (§5.2), and thus demonstrate that the model's performance is improving *for the right reasons*. Finally, we show that the proposed approach leads to better *compositional generalization* to out-of-distribution examples (§5.3). Our results show that we achieve the stated promise of latent decision models: an interpretable model that naturally encodes compositional reasoning and uses its modular architecture for better generalization.

2 Paired Examples as Indirect Supervision for Latent Decisions

We focus on structured compositional models for reasoning that perform an explicit problem decomposition and predict interpretable latent decisions that are composed to predict the final output. These intermediate outputs are often grounded in real-world phenomena and provide some explanation for the model's predictions. Such models assume that the structured architecture provides a useful inductive bias for efficient learning. For example, for a given input x, the computation performed to predict the output y can be expressed as a computation tree,

$$y = f(g(x), h(x)) \tag{1}$$

where, f, g, h perform the three sub-tasks required for x and the outputs of g and h are the intermediate decisions. The actual computation tree would be dependent on the input and the structure of the model. For example, to answer *How many field goals were scored?*, a NMN would perform $y = f(g(x))$ where $g(x)$ would output the set of *field goals* and f would return the size of this set. While we focus on NMNs in this paper, other models that have similar structures where our techniques would be applicable include language models with latent variables for coreference (Ji et al., 2017), syntax trees (Dyer et al., 2016), or knowledge graphs (Logan et al., 2019); checklist-style models that manage coverage over parts of the input (Kiddon et al., 2016); or any neural model that has some interpretable intermediate decision, including standard attention mechanisms (Bahdanau et al., 2015).

Typically, the only supervision provided to the model are gold (x, y) pairs from which it is expected to jointly learn the parameters of all of its components. Such weak supervision is not enough for accurate learning, and the fact that incorrect latent decisions can lead to the correct prediction further complicates learning. Consequently, models fail to learn to perform these latent tasks correctly and usually end up modeling irrelevant correlations in the data (Johnson, 2007; Subramanian et al., 2020).

In this work, we propose a method to leverage *paired examples*—examples whose one or more latent decisions are related to each other—to provide an indirect supervision to these latent decisions. Consider paired training examples x_i and x_j with computation trees,

$$y_i = f(g(x_i), h(x_i)) \tag{2}$$

$$y_j = f(k(g(x_j))) \tag{3}$$

that share the internal substructure $g(x)$. In such a scenario, we propose an additional training objective $S(g(x_i), g(x_j))$ to enforce consistency of partial model execution for the shared substructure:

$$\mathcal{L}_{\text{paired}} = S(g(x_i), g(x_j)) \tag{4}$$

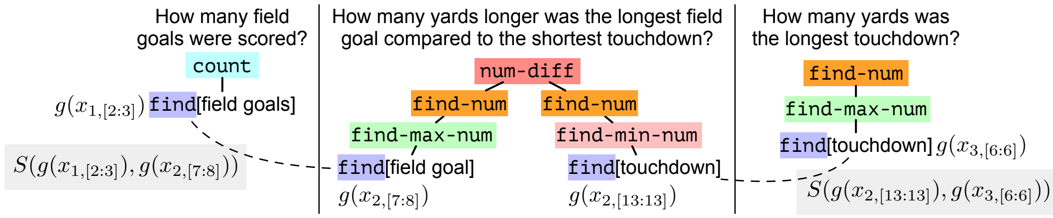

Figure 1: **Proposed paired objective:** For training examples that share substructure, we propose an additional training objective relating their latent decisions; S in the shaded gray area. In this figure, since the outputs of the substructure should be the same, S would encourage *equality* between them.

For example, in Figure 1, where the intermediate outputs $g(x)$ should be the same for the paired examples, using a similarity measure for S would enforce equality of the latent outputs. By adding this consistency objective, we are able to provide an additional training signal to the latent decision using related examples, and hence indirectly share supervision among multiple training examples. As a result, we are able to more densely characterize the decision boundary around an instance (x_i), by using related instances (x_j), than what was possible by using the original instance alone.

To use this consistency objective for x_i, we do not require supervision for the latent output $g(x_i)$, nor the gold end-task output y_j^* for the paired example x_j; we only enforce that the intermediate decisions are consistent. Additionally, we are not limited to enforcing consistency for a single intermediate decision from a single paired example; if x_i shares an additional substructure $h(x)$ with a paired example x_k, we can add an additional term $S'(h(x_i), h(x_k))$ to Eq. 4. Finally, different consistency relations can be enforced using different objectives.

Our approach generalizes a few previous methods for learning via paired examples. For learning to ground tokens to image regions, Gupta et al. (2020b) enforce contrastive grounding between the original and a negative token; this is equivalent to using an appropriate S in our framework. A few approaches (Minervini & Riedel, 2018; Li et al., 2019; Asai & Hajishirzi, 2020) use an additional objective on model outputs to enforce domain-specific consistency between paired examples; this is a special case of our framework where S is used on the outputs (y_i, y_j), instead of the latent decisions.

3 Training via Paired Examples in Neural Module Networks

We apply our approach to improve question answering using a neural module network (NMN; Andreas et al., 2016) on the DROP dataset (Dua et al., 2019). DROP contains complex compositional questions against natural language passages describing football games and historical events.

NMN is a model architecture aimed at reasoning about natural language against a given context (text, image, etc.) in a compositional manner. A NMN maps the input utterance into an executable program representing the compositional reasoning structure required to predict the output. The program is composed of learnable modules that are designed to perform atomic reasoning tasks. For example, to answer $q =$ *How many field goals were scored in the first half?*, a NMN would parse it into a program $z =$ `count`(`filter`[*in the first half*](`find`[*field goals*])); essentially performing, $y = f(g(q, h(q)))$, where $f =$ `count`, $g =$ `filter`, and $h =$ `find`.

Given a question q, the gold program z^*, and the correct answer a^*, maximum likelihood training is used to jointly train the parameters of the modules, as well as a parser that produces the gold program z^*. It is challenging to learn the module parameters correctly only using the answer supervision, especially since the space of possible intermediate outputs is quite large. For example, the `find` module needs to learn to select the correct spans among all possible spans in the passage.

Text-NMN We work with the Text-NMN of Gupta et al. (2020a) on a subset of DROP which is annotated with gold programs. Their models contain `find`, `filter`, `project`, `count`, `find-num`, `find-date`, `find-max-num`, `find-min-num`, `num-compare`, `date-compare`, `num-add`, `num-diff`, `time-diff`, and `spans` modules. The `find`, `filter`, and `project` modules take as input an additional question string argument. Each module's output is an attention distribution over the relevant support. E.g. `find`, `filter`, `project` output an attention over passage tokens, `find-num`, `num-add` over numbers, `find-date` over dates etc. Please refer to their paper for details.

Paired training in NMNs We consider a pair of questions whose program trees share a subtree as paired examples. A shared subtree implies that a part of the reasoning required to answer the questions is the same. Since some modules take as input a string argument, we define two subtrees to be equivalent *iff* their structure matches and the string arguments to the modules that require them are *semantically equivalent*. For example, subtrees `find-num`(`find`[*passing touchdowns*]) and `find-num`(`find`[*touchdown passes*]) are equivalent, while they are not the same as `find-num`(`find`[*touchdown runs*]) (we describe how we detect semantic equivalence in §4).

Consider a question q_i that shares the substructure $g(q)$ with a paired question q_j. Since shared substructures are common program subtrees in our case, we encourage the latent decisions to be equal. As the outputs of modules are probability distributions, we minimize the KL-divergence between the two outputs to enforce consistency. We maximize the following paired objective from Eq. 4,

$$\mathcal{L}_{\text{paired}} = -\big(\text{KL}[g(q_i) \parallel g(q_j)] + \text{KL}[g(q_j) \parallel g(q_i)]\big) \tag{5}$$

where $S(p_1, p_2) = -(\text{KL}[p_1 \parallel p_2] + \text{KL}[p_2 \parallel p_1])$ is the negative symmetric KL-divergence.

Complete Example We describe the benefits of training using paired examples using an example. Consider the four questions in Figure 2; all of them share the substructure of finding the field goal scoring events. However, we find that for the questions requiring the `find-{max/min}-num` operation, a vanilla NMN directly grounds to the longest/shortest field goal as the `find` execution. Due to the use of powerful NNs for contextualized question/passage representations (e.g. BERT) and no constraints on the modules to perform as intended, the model performs the symbolic *min*/*max* operation internally in its parameters. Such `find` execution results in non-interpretable behavior, and substantially hurts generalization to the count questions. By enforcing consistency between all the `find` executions, the model can no longer shortcut the compositional reasoning defined by the programs; this results in correct `find` outputs and better generalization, as we show in §5.4.

4 Many Ways of Getting Paired Data

We explore three ways of acquiring paired questions. We show how questions that share substructures can be automatically found from within the dataset (§4.1), and how new paired questions can be constructed using templates (§4.2.1), or generated using a question-generation model (§4.2.2).

4.1 Finding Naturally Occurring Paired Data

Any dataset that contains multiple questions against the same context could have questions that query different aspects of the same underlying event or entity. These examples can potentially be paired by finding the elements in common between them. As the DROP data that we are using has annotated programs, this process is simplified somewhat in that we can simply find pairs of programs in the training data that share a subtree. While the subtrees could be of arbitrary size, we limit ourselves to programs that share a leaf `find` module. Recall that `find` requires a question string argument, so the challenge of finding paired questions reduces to discovering pairs of `find` modules in different questions about the same paragraph whose question string arguments are semantically equivalent. To this end, we use BERTScore (Zhang* et al., 2020) to measure string similarity.

We consider two string arguments to be semantically equivalent if their BERTScore-F1 exceeds a threshold (0.6), and if the same entities are mentioned in the arguments. This additional constraint allows us to judge that *Jay Feely's field goal* and *Janikowski's field goal* are semantically different, even though they receive a high BERTScore. This approach would find paired examples like,
What term is used to describe the Yorkist defeat at Ludford Bridge in 1459?
What happened first: Yorkist defeat at Ludford Bridge or widespread pillaging by Queen Margaret?

4.2 Paired Data via Augmentation

One benefit of our consistency objective (Eq. 4) is that it only requires that the paired example shares substructure. This allows us to augment training data with new paired questions without knowing their gold answer. We explore two ways to carry out this augmentation; (a) constructing paired questions using templates, and (b) generating paired questions using a question-generation model.

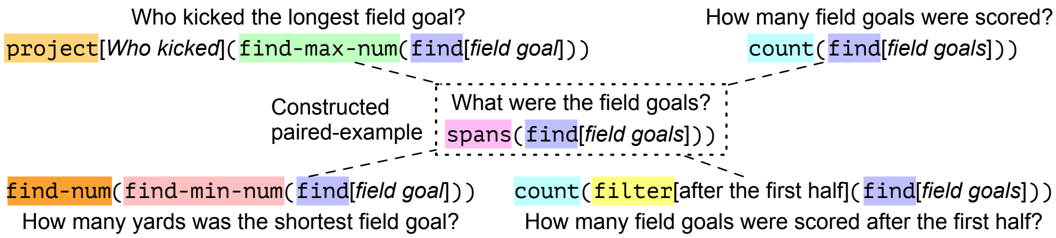

Figure 2: **Templated Construction of Paired Examples**: Constructed paired examples can help in indirectly enforcing consistency between different training examples (§4.2.1).

4.2.1 Templated Construction of Paired Examples

Grounding `find` event(s) Using the question argument from the `find` module of certain frequently occurring programs, we construct a paired question that aims to ground the mentions of the event queried in the `find` module. For example, *Who scored the longest touchdown?* would be paired with *What were the touchdowns?*. This templated paired question construction is carried out for, (1) `count(find[])` (2) `count(filter(find[]))` (3) `find-num(find-max-num(find[]))` (4) `find-num(find-max-num(filter[](find[])))` (5) `project[](find-max-num(find[]))` (6) `project[](find-max-num(filter[](find[])))` (7) `date-compare-gt(find[], find[])` (8) `time-diff(find[], find[])`, and their versions with `find-min-num` or `date-compare-lt`.

For questions with a program in (1) - (6), we append *What were the* to the program's `find` argument to construct a paired question. We annotate this paired question with the program `spans(find[])`, and enforce consistency among the `find` modules. Such a construction allows us to indirectly enforce consistency among multiple related questions via the constructed question; see Figure 2.

For questions with a program in (7) - (8), we append *When did the* to the two `find` modules' arguments and construct two paired questions, one for each `find` operation. We label the constructions with `find-date(find[])` and enforce consistency among the `find` modules. For example, *How many years after the Battle of Rullion Green was the Battle of Drumclog?* would result in the construction of *When did the Battle of Rullion Green?* and *When did the Battle of Drumclog?*. While this method can lead to ungrammatical questions, it should help in decomposing the two `find` executions.

Inverting Superlatives For questions with a program in (3) - (6) or its `find-min-num` equivalent, we construct a paired question by replacing the superlative in the question with its antonym (e.g. *largest* $\rightarrow$ *smallest*) and inverting the min/max module. We enforce consistency among the `find` modules of the original and the paired question.

4.2.2 Model-generated Paired Examples

We show how question generation (QG) models (Du et al., 2017; Krishna & Iyyer, 2019) can be used to generate paired questions. QG models are seq2seq models that generate a question corresponding to an answer span marked in a passage as input. We follow Wang et al. (2020) and fine-tune a BART model (Lewis et al., 2020) on SQuAD (Rajpurkar et al., 2016) to use as a QG model.

We generate paired questions for non-football passages[1] in DROP by randomly choosing 10 numbers and dates as answer spans, and generating questions for them. We assume that the generated questions are SQuAD-like—they query an argument about an event/entity mentioned in text—and label them with the program `find-num(find)` or `find-date(find)`. We then follow the same procedure as §4.1—for each of the `find` module in a DROP question's program, we see if a *semantically similar* generated question exists. If such an augmented question is found, it is used as a paired example for the DROP question to enforce consistency between the `find` modules. For example, *How many percentage points did the population of non-Hispanic Whites drop from 1990 to 2010?* is paired with the generated question *What percentage of the population was non-Hispanic Whites in 2010?*.

[1]We explain the reason for this in §A.2

5 Experiments

Dataset and Setup We perform experiments on the subset of the DROP dataset (Dua et al., 2019) that is covered by the modules in Text-NMN. This subset is a union of the data used by Gupta et al. (2020a) and the question decomposition annotations in the BREAK dataset (Wolfson et al., 2020). All questions in our dataset contain program annotations (heuristically annotated by Gupta et al. (2020a); crowd-sourced in BREAK). We only use these program annotations for training; all validation and test results are based on predicted programs. Our complete subset of DROP contains 23215 question-answer pairs. For an i.i.d. split, since the DROP test set is hidden, we split the training set into train/validation and use the provided validation set as the test set. Our train / validation / test sets contain 18299 / 2460 / 2456 questions, respectively. In the training data, we found 7018 naturally-occurring pairings for 6039 questions (§4.1); construct template-based paired examples for 10882 questions (§4.2.1); and generate 2632 questions paired with 2079 DROP questions (§4.2.2).

Baselines As we are studying the impact of our new paired learning objective, our main point of comparison is a Text-NMN trained without that objective. Though the focus of our work is structured interpretable models, we also show results from a strong, reasonably comparable black-box model for DROP, MTMSN (Hu et al., 2019), to better situate the relative performance of this class of models.

Hyperparameters and other experimental details are described in §A.1. We release all our data and code publicly at `http://omitted.link`.

5.1 In-distribution Performance

We first evaluate the impact of our proposed paired objective on in-distribution generalization. Table 1 shows the performance of the NMNs, trained with and without the paired objective, using different types of paired examples. We see that paired objective always leads to improved performance; test F1 improves from 70.3 F1 for the vanilla NMN to (a) 71 F1 using naturally-occurring paired examples ($\mathcal{L}_{\text{paired, found}}$), (b) 72.3 F1 using template-based paired examples ($\mathcal{L}_{\text{paired, temp}}$), and (c) 71.2 F1 using model-generated paired examples ($\mathcal{L}_{\text{paired, qgen}}$). Further, the model achieves the best performance when all kinds of paired examples are combined, improving the performance to 73.5 F1 ($\mathcal{L}_{\text{paired, all}}$).[2] Our final model also outperforms the black-box MTMSN model.

Model	dev		test	
	F1	**EM**	**F1**	**EM**
MTMSN	66.2	62.4	72.8	70.3
NMN Baseline	62.6	58.0	70.3	67.0
NMN + $\mathcal{L}_{\text{paired, found}}$	66.0	61.5	71.0	67.8
NMN + $\mathcal{L}_{\text{paired, temp}}$	66.2	61.4	72.3	69.2
NMN + $\mathcal{L}_{\text{paired, qgen}}$	63.7	58.9	71.2	68.4
NMN + $\mathcal{L}_{\text{paired, all}}$	66.3	**61.6**	**73.5**	**70.5**

Table 1: **Performance on DROP (pruned):** Using our paired objective with all different kinds of paired-data leads to improvements in NMN. Model achieves the best performance when all kinds of paired-data are used together.

Model	Performance (F1 Score)	Overall Faithful. (cross-entropy* ↓)	Module-wise Faithfulness* (↓)				
			`find`	`filter`	`num-date`†	`project`	`min-max`†
NMN	70.3	46.3	14.3	21.0	30.6	0.9	1.4
NMN + $\mathcal{L}_{\text{paired, all}}$	73.5	13.0	4.4	5.7	8.3	1.4	1.2

Table 2: **Faithfulness scores**: Using the paired objective significantly improves intermediate output predictions. †denotes the average of `find-num` & `find-date` and `find-min-num` & `find-max-num`.

5.2 Measuring faithfulness of NMN Execution

As observed by Subramanian et al. (2020), training a NMN only using the end-task supervision can lead to learned modules whose behaviour is *unfaithful* to their intended reasoning operation, even when trained and evaluated with gold programs. That is, even though the NMN might produce the correct final output, the outputs of the modules are not as expected according to the program (e.g., outputting only the longest field goal for the `find`[*field goal*] execution), and this leads to markedly

[2]The improvement over the baseline is statistically significant (p = 0.01) based on the Student's t-test. Test numbers are much higher than dev since the test set contains 5 answer annotations for each question.

Model	Complex Arithmetic			Filter-ArgMax		
	dev	**test w/o G.P.**	**test w/ G.P.**	**dev**	**test w/o G.P.**	**test w/ G.P.**
MTMSN	67.3	44.1		67.5	59.3	
NMN	64.3	29.5	42.1	65.0	55.6	59.7
NMN + $\mathcal{L}_{\text{paired, all}}$	67.2	47.2	**54.7**	65.5	62.3	**71.5**

Table 3: **Measuring compositional-generalization**: NMN performs substantially better when trained with the paired objective and performs even better when gold-programs are used (w/ G.P).

worse generalization on DROP. They release annotations for DROP containing the correct spans that should be output by each module in a program, and propose a cross-entropy-based metric to quantify the divergence between the output distribution over passage tokens and the annotated spans. A lower value of this metric denotes better faithfulness of the produced outputs.

We evaluate whether the use of our paired objective to indirectly supervise latent decisions (module outputs) in a NMN indeed leads to more faithful execution. In Table 2 we see that the NMN trained with the proposed paired objective greatly improves the overall faithfulness ($46.3 \rightarrow 13.0$) and also leads to huge improvements in most modules. This faithfulness evaluation shows that enforcing consistency between shared substructures provides the model with a dense enough training signal to learn correct module execution. That is, not only does the model performance improve by using the paired objective, this faithfulness evaluation shows that the model's performance is improving *for the right reasons*. In §5.4 we explore how this faithfulness is actually achieved.

5.3 Evaluating Compositional Generalization

A natural expectation from structured models is that the explicit structure should help the model learn *reusable* operations that generalize to novel contexts. We test this capability using the *compositional generalization* setup of Finegan-Dollak et al. (2018), where the model is tested on questions whose program templates are unseen during training. In our case, this tests whether module executions generalize to new contexts in a program.

We create two test sets to measure our model's capability to generalize to such out-of-distribution examples. In both settings, we identify certain program templates to keep in a held-out test set, and use the remaining questions for training and validation purposes.

Complex Arithmetic This test set contains questions that require add/sub operations in complex contexts; questions whose program contains the `num-add`/`num-diff` as the root node, but the program is *not* the simple add/sub template `num-add`/`num-diff(find-num(find), find-num(find))`. For example, *How many more mg/L is the highest amount of arsenic in drinking water linked to skin cancer risk than the lowest mg/L amount?*, with program `num-diff(find-num(find-max-num(find)), find-num(find-min-num(find)))`.

Filter-Argmax This test set contains questions that require an argmax operation after filter; programs that contain the `find-max-num`/`find-min-num(filter(·))` subtree. For example, *Who scored the shortest touchdown in the first half?*, with program `project(find-max-num(filter(find)))`.

Performance In Table 3 we see that a NMN using our paired objective outperforms both the vanilla NMN and the black-box MTMSN on both test sets.[3] This shows that enforcing consistent module behavior also improves their performance in novel contexts and as a result allows the model to generalize to out-of-distribution examples. We see a further dramatic improvement in performance when the model is evaluated using gold programs. This is not surprising since it is known that semantic parsers (including the one in our model) often fail to generalize compositionally (Finegan-Dollak et al., 2018; Lake & Baroni, 2018; Bahdanau et al., 2019). Recent advancements in semantic parsing models that aim at compositional generalization should help improve overall model performance (Lake, 2019; Korrel et al., 2019; Herzig & Berant, 2020).

[3]The test set size is quite small, so while the w/ G.P. results are significantly better than MTMSN (p = 0.05), we can't completely rule out noise as the cause for the w/o G.P. result (p = 0.5), based on the Student's t-test.

5.4 ANALYSIS

Model	Test F1			Faithful.-score (↓)
	Overall	Min-Max	Count	
NMN	57.4	82.1	36.2	110.4
+ $\mathcal{L}_{\text{max+min}}$	60.9	**85.5**	39.7	56.5
+ $\mathcal{L}_{\text{max+count}}$	60.8	81.4	43.0	99.2
+ $\mathcal{L}_{\text{max+min+count}}$	**71.1**	85.4	**58.8**	**25.9**

Table 4: Using constructed paired examples for all three types of questions—min, max, and count—leads to dramatically better count performance. Without all three, the model finds shortcuts to satisfy the consistency constraint and does not learn correct module execution.

We perform an analysis to understand how augmented paired examples—ones that do not contain end-task supervision—help in improving latent decision predictions. We conduct an experiment on a subset of the data containing only min, max and count type questions; programs in (1)-(6) from §4.2.1. We see a dramatic improvement over the baseline in count-type performance when paired examples for all three types of questions are used; answer-F1 improves from $36.2 \rightarrow 58.8$, and faithfulness from $110.4 \rightarrow 25.9$. This verifies that without additional supervision the model does indeed perform the min/max operation internal to its parameters and ground to the output event instead of performing the correct `find` operation (§3). As a result, the `find` computation that *should* be shared with the count questions is not actually shared, hurting performance. By indirectly constraining the `find` execution to produce consistent outputs for all three types of questions, via the constructed question (Fig. 2), the model learns to correctly execute `find`, resulting in much better count performance. Using paired examples only for max and count questions ($\mathcal{L}_{\text{max+count}}$) does not constrain the `find` operation sufficiently—the model has freedom to optimize the paired objective by learning to incorrectly ground to the max-event mention for both the original and constructed question's `find` operation. This analysis reveals that augmented paired examples are only useful if they form enough indirect connections between different types of instances, that is sufficient to densely characterize the decision boundary around the latent decisions.

6 RELATED WORK

The challenge in learning models for complex problems can be viewed as the emergence of artificially simple decision boundaries due to data sparsity and presence of spurious dataset biases (Gardner et al., 2020). To counter data sparsity, data augmentation techniques have been proposed to provide a compositional inductive bias to the model (Chen et al., 2020; Andreas, 2020) or induce consistent outputs (Asai & Hajishirzi, 2020; Ribeiro et al., 2019). However, their applicability is limited to problems where the end-task supervision (y) for the augmented examples can be easily inferred. To counter dataset biases, model-based data pruning (AFLite; Bras et al., 2020) and subsampling (Oren et al., 2020) have been proposed. All the techniques above modify the training-data distribution to remove a model's propensity to find artificially simple decision boundaries, whereas we modify the training objective to try to accomplish the same goal. Ensemble-based training methodology (Clark et al., 2019; Stacey et al., 2020) has been proposed to learn models robust to dataset artifacts; however, they require prior knowledge about the kind of artifacts present in the data. Our approach, in spirit, is related to a large body of work on learning structured latent variable models. For example, prior work has incorporated indirect supervision via constraints (Graça et al., 2007; Chang et al., 2007; Ganchev et al., 2010) or used negative examples with implausible latent structures (Smith & Eisner, 2005; Chang et al., 2010). These approaches use auxiliary objectives on a single training instance or global conditions on posterior distributions, whereas our training objective uses *paired examples*.

7 CONCLUSION

We propose a method to leverage *paired examples*—instances that share internal substructure—to provide a richer training signal to latent decisions in compositional model architectures. We explore three methods to acquire paired examples and empirically show that our approach leads to substantially better in- and out-of-distribution generalization of a neural module network in complex compositional question answering. We also show that using our paired objective leads to improved prediction of latent decisions. A lot of recent work is exploring the use of closely related instances for improved evaluation and training. Ours is one of the first works to show dramatic improvements by modifying the training objective to try to make better use of the local decision surface. These results should encourage more work exploring this direction.

REFERENCES

Jacob Andreas. Good-enough compositional data augmentation. In *ACL*, 2020.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *CVPR*, 2016.

Akari Asai and Hannaneh Hajishirzi. Logic-guided data augmentation and regularization for consistent question answering. In *ACL*, 2020.

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

Dzmitry Bahdanau, H. D. Vries, Timothy J. O'Donnell, Shikhar Murty, Philippe Beaudoin, Yoshua Bengio, and Aaron C. Courville. CLOSURE: Assessing Systematic Generalization of CLEVR Models. In *ViGIL@NeurIPS*, 2019.

Ronan Le Bras, Swabha Swayamdipta, Chandra Bhagavatula, Rowan Zellers, Matthew E. Peters, A. Sabharwal, and Yejin Choi. Adversarial filters of dataset biases. In *ICML*, 2020.

Ming-Wei Chang, Lev Ratinov, and Dan Roth. Guiding semi-supervision with constraint-driven learning. In *ACL*, 2007.

Ming-Wei Chang, V. Srikumar, Dan Goldwasser, and D. Roth. Structured output learning with indirect supervision. In *ICML*, 2010.

Xinyun Chen, Chen Liang, Adams Wei Yu, Denny Zhou, D. Song, and Quoc V. Le. Neural symbolic reader: Scalable integration of distributed and symbolic representations for reading comprehension. In *ICLR*, 2020.

Christopher Clark, Mark Yatskar, and Luke Zettlemoyer. Don't take the easy way out: Ensemble based methods for avoiding known dataset biases. In *EMNLP*, 2019.

X. Du, Junru Shao, and Claire Cardie. Learning to ask: Neural question generation for reading comprehension. In *ACL*, 2017.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *NAACL-HLT*, 2019.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. Recurrent neural network grammars. In *HLT-NAACL*, 2016.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. Improving text-to-sql evaluation methodology. In *ACL*, 2018. URL `https://www.aclweb.org/anthology/P18-1033`.

Kuzman Ganchev, Joao Graça, Jennifer Gillenwater, and Ben Taskar. Posterior regularization for structured latent variable models. *The Journal of Machine Learning Research*, 2010.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew E. Peters, Michael Schmitz, and Luke S. Zettlemoyer. Allennlp: A deep semantic natural language processing platform. *ArXiv*, abs/1803.07640, 2018.

Matt Gardner, Yoav Artzi, Victoria Basmova, Jonathan Berant, Ben Bogin, Sihao Chen, Pradeep Dasigi, Dheeru Dua, Yanai Elazar, Ananth Gottumukkala, Nitish Gupta, Hanna Hajishirzi, Gabriel Ilharco, Daniel Khashabi, Kevin Lin, Jiangming Liu, Nelson F. Liu, Phoebe Mulcaire, Qiang Ning, Sameer Singh, Noah A. Smith, Sanjay Subramanian, Reut Tsarfaty, Eric Wallace, A. Zhang, and Ben Zhou. Evaluating models' local decision boundaries via contrast sets. In *Findings of EMNLP*, 2020.

J. Graça, K. Ganchev, and B. Taskar. Expectation maximization and posterior constraints. In *NIPS*, 2007.

Nitish Gupta, Kevin Lin, Dan Roth, Sameer Singh, and Matt Gardner. Neural module networks for reasoning over text. In *ICLR*, 2020a. URL `https://arxiv.org/abs/1912.04971`.

Tanmay Gupta, Arash Vahdat, Gal Chechik, Xiaodong Yang, Jan Kautz, and Derek Hoiem. Contrastive learning for weakly supervised phrase grounding. *ECCV*, 2020b.

Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel R. Bowman, and Noah A. Smith. Annotation artifacts in natural language inference data. In *NAACL-HLT*, 2018.

Jonathan Herzig and Jonathan Berant. Span-based semantic parsing for compositional generalization. *ArXiv*, abs/2009.06040, 2020.

Minghao Hu, Yuxing Peng, Zhiheng Huang, and Dongsheng Li. A multi-type multi-span network for reading comprehension that requires discrete reasoning. In *EMNLP*, 2019.

Yangfeng Ji, Chenhao Tan, Sebastian Martschat, Yejin Choi, and Noah A. Smith. Dynamic entity representations in neural language models. In *EMNLP*, 2017.

Mark Johnson. Why doesn't em find good hmm pos-taggers? In *EMNLP-CoNLL*, 2007.

Chloé Kiddon, Luke Zettlemoyer, and Yejin Choi. Globally coherent text generation with neural checklist models. In *EMNLP*, 2016.

K. Korrel, Dieuwke Hupkes, Verna Dankers, and Elia Bruni. Transcoding compositionally: using attention to find more generalizable solutions. In *ACL*, 2019.

Kalpesh Krishna and Mohit Iyyer. Generating question-answer hierarchies. In *ACL*, 2019.

A. Lai and Julia Hockenmaier. Illinois-lh: A denotational and distributional approach to semantics. In *SemEval@COLING*, 2014.

B. M. Lake. Compositional generalization through meta sequence-to-sequence learning. In *NeurIPS*, 2019.

B. M. Lake and M. Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *ICML*, 2018.

M. Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, A. Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *ACL*, 2020.

Tao Li, Vivek Gupta, Maitrey Mehta, and V. Srikumar. A logic-driven framework for consistency of neural models. In *EMNLP/IJCNLP*, 2019.

IV Robert L. Logan, Nelson F. Liu, Matthew E. Peters, Matt Gardner, and Sameer Singh. Barack's wife hillary: Using knowledge-graphs for fact-aware language modeling. In *ACL*, 2019.

Sewon Min, Eric Wallace, Sameer Singh, Matt Gardner, Hannaneh Hajishirzi, and Luke Zettlemoyer. Compositional questions do not necessitate multi-hop reasoning. In *ACL*, 2019.

Pasquale Minervini and S. Riedel. Adversarially regularising neural nli models to integrate logical background knowledge. In *CoNLL*, 2018.

Inbar Oren, Jonathan Herzig, Nitish Gupta, Matt Gardner, and Jonathan Berant. Improving compositional generalization in semantic parsing. In *Findings of EMNLP*, 2020.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. In *EMNLP*, 2016.

Marco Tulio Ribeiro, Carlos Guestrin, and Sameer Singh. Are red roses red? evaluating consistency of question-answering models. In *ACL*, 2019.

Noah A. Smith and J. Eisner. Contrastive estimation: Training log-linear models on unlabeled data. In *ACL*, 2005.

Joe Stacey, Pasquale Minervini, Haim Dubossarsky, Sebastian Riedel, and Tim Rocktäschel. There is strength in numbers: Avoiding the hypothesis-only bias in natural language inference via ensemble adversarial training. *ArXiv*, abs/2004.07790, 2020.

Sanjay Subramanian, Ben Bogin, Nitish Gupta, Tomer Wolfson, Sameer Singh, Jonathan Berant, and Matt Gardner. Obtaining Faithful Interpretations from Compositional Neural Networks. In *ACL*, 2020. URL `https://www.aclweb.org/anthology/2020.acl-main.495`.

Alex Wang, Kyunghyun Cho, and M. Lewis. Asking and answering questions to evaluate the factual consistency of summaries. In *ACL*, 2020.

Tomer Wolfson, Mor Geva, Ankit Gupta, Matt Gardner, Yoav Goldberg, Daniel Deutch, and Jonathan Berant. Break it down: A question understanding benchmark. *TACL*, 2020.

Tianyi Zhang*, Varsha Kishore*, Felix Wu*, Kilian Q. Weinberger, and Yoav Artzi. BERTScore: Evaluating Text Generation with BERT. In *ICLR*, 2020. URL `https://openreview.net/forum?id=SkeHuCVFDr`.

A Appendix

A.1 Experimental Details

All models use the bert-base-uncased model to compute the question and passage contextualized representations. For all experiments, we train two versions of the model with different seed values, and choose the one that results in higher validation performance. All models are trained for a maximum number of 40 epochs, with early stopping if validation F1 does not improve for 10 consecutive epochs. For MTMSN, we use the hyperparameters provided with the original code. For NMN, we use a batch size of 2 (constrained by a 12GB GPU) and a learning rate of `1e-5`. The question-parser in the Text-NMN uses a 100-dimensional, single-layer LSTM decoder.

Our code is written using the AllenNLP library (Gardner et al., 2018).

Dataset As mentioned in §5, our dataset is composed of two subsets of DROP; (1) The subset of DROP used in Gupta et al. (2020a)—this contains 3881 passages and 19204 questions, and (2) question-decomposition meaning representation (QDMR) annotations from Break (Wolfson et al., 2020)—this contains 2756 passages with a total of 4762 questions. After removing duplicate questions we are left with 23215 questions in total. We convert the program annotations in QDMR to programs that conform to the grammar induced by the modules in Text-NMN using simple transformations.

We will publicly release all our code, data, and trained-model checkpoints for reproducibility.

A.2 Model-generated Paired Examples

For the question generation model we use the BART-large model and train for 1 epoch using a learning rate of `3e-5`. From the SQuAD dataset, we as training data only questions whose answer text appears exactly once in the passage.

As mentioned in §4.2.2, we only generate paired questions for non-football questions based on two frequent observations, both related to domain shift. Consider this snippet from a passage containing a football game summary:

Following their road loss to the Steelers, the Browns flew to M&T Bank Stadium for an AFC North rematch with the Baltimore Ravens. ..... the Browns showed signs of life as QB Derek Anderson completed a 3-yard TD pass to WR Joe Jurevicius. ..... Cleveland tied the game at 17-17 with Anderson's 14-yard TD pass to WR Braylon Edwards. ..... the Ravens took over for the rest of the game with Boller's 77-yard TD pass to WR Demetrius Williams.

If we generate a question conditioned on the number 3 as the answer, our QG model typically generates a question such as *How many yards was the TD pass?* or *How many yards Anderson's pass?*. Both these questions are *not correct*, as the answer to them is not just the conditioned answer

(3), as there are multiple possible answers in the given paragraph. In the football game summary domain, it is common for a single event type to contain multiple mentions like this, which our QG model trained on SQuAD cannot handle. Similarly, we observe that the QG model generates nonsensical questions for numbers associated to game scores (e.g. 17-17), likely due to domain shift. Future work should look into QG models that can operate under different domains to generate paired examples.