Weakly Supervised Neuro-Symbolic Module Networks for Numerical Reasoning over Text

Amrita Saha,[1] **Shafiq Joty,** [1,2] **Steven C.H. Hoi** [1]

[1] Salesforce Research Asia
[2] Nanyang Technological University
amrita.saha, sjoty, shoi@salesforce.com

Abstract

Neural Module Networks (NMNs) have been quite successful in incorporating explicit reasoning as learnable modules in various question answering tasks, including the most generic form of numerical reasoning over text in Machine Reading Comprehension (MRC). However to achieve this, contemporary Neural Module Networks models obtain strong supervision in form of specialized program annotation from the QA pairs through various heuristic parsing and exhaustive computation of all possible discrete operations on discrete arguments. Consequently they fail to generalize to more open-ended settings without such supervision. Hence, we propose **W**eakly-Supervised **N**euro-**S**ymbolic **M**odule **N**etwork (WNSMN) trained with answers as the sole supervision for numerical reasoning based MRC. WNSMN learns to execute a noisy heuristic program obtained from the dependency parse of the query, as discrete actions over both neural and symbolic reasoning modules and trains it end-to-end in a reinforcement learning framework with discrete reward from answer matching. On the subset of DROP having numerical answers, WNSMN outperforms NMN by 32% and the reasoning-free generative language model GenBERT by 8% in exact match accuracy under comparable weakly supervised settings. This showcases the effectiveness of modular networks that can handle explicit discrete reasoning over noisy programs in an end-to-end manner.

Introduction

End-to-end neural models have proven to be powerful tools for an expansive set of language and vision problems by effectively emulating the *input-output* behavior. However, many real problems like Question Answering (QA) or Dialog need more interpretable models that can incorporate explicit reasoning in the inference process. In this work, we focus on the most generic form of numerical reasoning over text, encompassed by the reasoning-based MRC framework. A particularly challenging setting for this task is where the answers are numerical in nature as in the popular MRC dataset, DROP (Dua et al. 2019). The example in Figure 1 shows the intricacies involved in the task which include (*i*) passage and query language understanding, (*ii*) contextual understanding of the passage entities (dates, numbers), and (*iii*) application of quantitative reasoning (*e.g., max, not*) over those entities to reach the final numerical answer.

Three broad genres of models have proven successful on the DROP numerical reasoning task. First, large-scale *pretrained language models* like GenBERT (Geva, Gupta, and Berant 2020) use a monolithic Transformer architecture and decodes numerical answers digit-by-digit. Though they deliver mediocre performance when trained only on the target data, their competency is derived from pretraining on massive synthetic data augmented with explicit supervision of the gold numerical reasoning. Second kind of models are the *reasoning-free hybrid models* like NumNet (Ran et al. 2019), NAQANet (Dua et al. 2019), NABERT+ (Kinley and Lin 2019) and MTMSN (Hu et al. 2019), NeRd (Chen et al. 2020). They explicitly incorporate numerical computations in the standard extractive QA pipeline by learning a multi-type answer predictor over different reasoning types (*e.g., max/min*, *diff/sum*, *count*, *negate*) and directly predicting the corresponding numerical expression, instead of learning to reason. This is facilitated by exhaustively precomputing all possible outcomes of discrete operations and augmenting the training data with the reasoning-type supervision and numerical expressions that lead to the correct answer.

Lastly, the most relevant class of models for this work are the *modular networks for reasoning*. Neural Module Networks (NMN) (Gupta et al. 2020) is the first explicit reasoning based QA model which parses the query into a specialized program and executes it step-wise over learnable reasoning modules. However, to do so, apart from the exhaustive precomputation of all discrete operations, it also needs more fine-grained supervision of the gold program and the gold program execution, obtained heuristically, by leveraging the abundance of templatized queries in DROP. While being more pragmatic and richer at interpretability, both modular and hybrid networks are also tightly coupled with the additional supervision *i.e.,* they cannot learn without it. While NMN is the first to *enable* learning from QA pair alone, it still needs more finer-grained supervision for at least a part of the training data. With this, it manages to supersede NABERT and MTMSN on a carefully chosen subset of DROP using the supervision, but generalizes poorly to more open-ended settings lacking such supervision.

Need for symbolic reasoning. One striking characteristic of the modular methods is to avoid discrete reason-

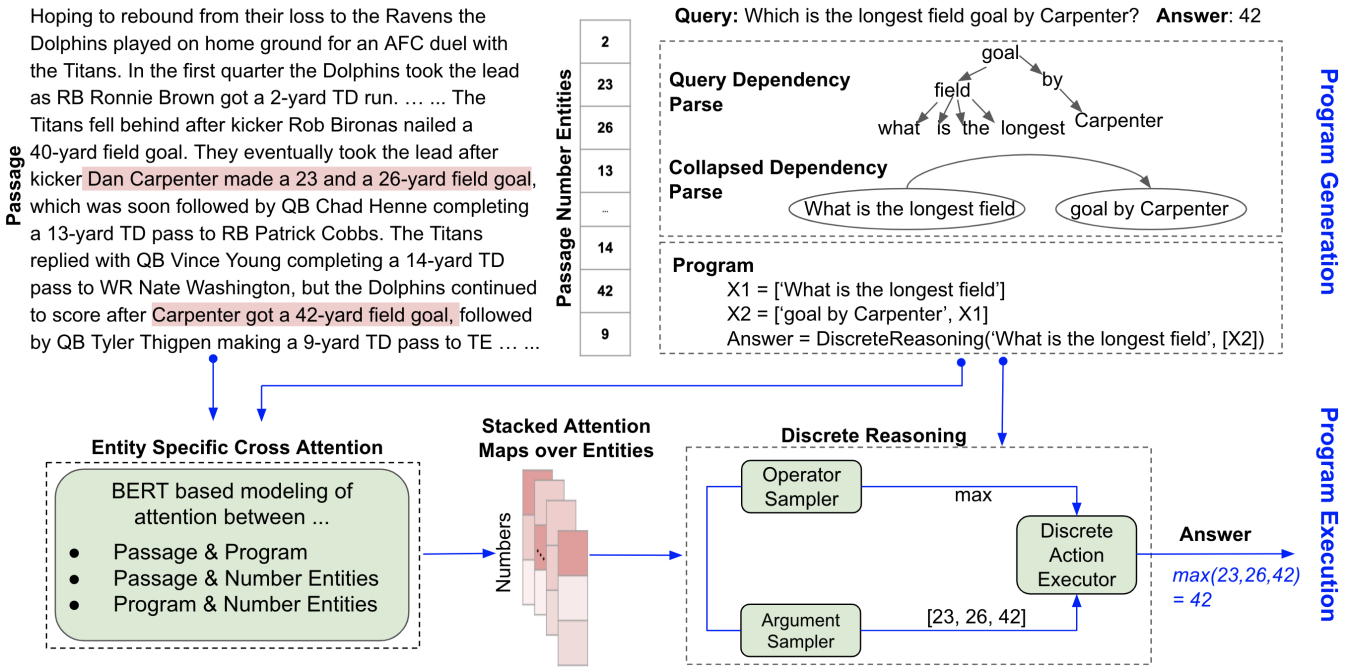


Figure 1: Outline of our method on example (passage, query, answer) from DROP: Executing noisy program obtained from dependency parsing of the query by learning entity specific cross attentions, and executing discrete operations on entity arguments to reach the answer.

ing by employing only learnable modules with an exhaustively precomputed space of outputs. While they perform well on DROP, their modeling complexity grows arbitrarily with complex non-linear numerical operations (*e.g.,* log, cos). Contrarily, symbolic modular networks that execute the discrete operations are possibly more robust or pragmatic in this respect by remaining unaffected by the operation complexity. Such discrete reasoning has indeed been incorporated for simpler, well-structured tasks like math word problems with a small context and few number entities like (Koncel-Kedziorski et al. 2016; Roy and Roth 2015) or QA over structured data (Knowledge Bases or Tables) (Zhong, Xiong, and Socher 2017; Liang et al. 2018; Saha et al. 2019; Neelakantan et al. 2017), with Deep Reinforcement Learning (RL) for end-to-end training. For these tasks the programs enjoy well-defined input arguments and output variables (with variable types as KB entity-type or relation) and program execution is entirely symbolic with no learning required. However the fuzzy reasoning in MRC requires more generic framework of learnable program execution over neural and symbolic modules with open-ended arguments and attention-maps as outputs. This makes the RL framework more challenging, owing to the action-space explosion and sparse, confounding rewards. To our knowledge RL has not been applied in this complex setting.

In view of this, we propose a **W**eakly-Supervised **N**euro-**S**ymbolic **M**odule **N**etwork **(WNSMN)**:

- Is a first attempt at numerical reasoning based MRC, trained with answers as sole supervision;
- Is based on a generalized framework of dependency parsing of queries into noisy programs (specifically the Stanford Parser), which only captures the dependency structure of the query terms;
- Learns to execute the noisy programs through explicit reasoning by sampling discrete operators and arguments from the passage context;
- Takes a pragmatic approach by using combination of neural and symbolic modules, in order to learn in the absence of any heuristic supervision,
- Uses a deep Reinforcement Learning framework for training the neuro-symbolic reasoning modules end-to-end with binary rewards from exact answer match.

In contrast, the earlier SoTA models NMN, MTMSN and NeRd obtain strong supervision in terms of specialized program annotation from the QA pairs through various heuristic parsing of the templatized DROP queries and through exhaustive computation of all possible discrete operations on discrete arguments. The latter alleviates any need for explicit discrete reasoning, even when the task demands.

To concretely compare our proposed WNSMN with contemporary NMN, consider the example in Figure 1. In comparison to our generalized query-parsing, NMN parses the query into a program form *(MAX(FILTER(FIND('Carpenter'), 'goal'))*, which is step-wise executed by different learnable modules with exhaustively precomputed output set. To train the network, it needs to augment the training data with annotations of i) *gold program*, ii) *gold query attention i.e.,* query segment to attend as program argument in each step and iii) *gold program execution i.e.,* the *exact* discrete operation and numerical expression (*i.e.,* the numerical operation and operands) that leads to the correct answer *e.g.,* the

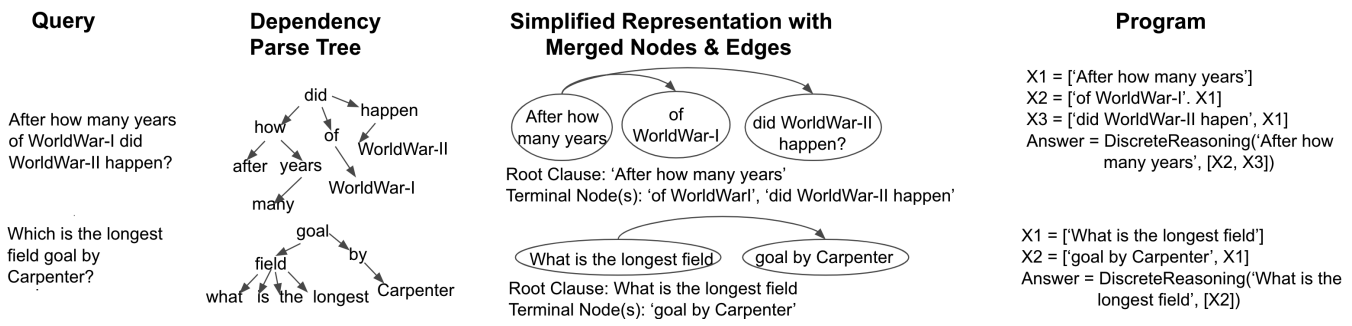


Figure 2: Examples of Programs for WNSMN obtained from the dependency parse of the query.

supervision of the gold numerical expression in Figure 1 is *MAX(23, 26, 42)*. These are obtained through manual inspection of the data through regex based pattern matching and heuristics applied on the query language. Such dependency on strong supervision restricts the applicability of NMN to a small subset of DROP where the queries match the desired patterns. Further, because of the abundance of templatized queries in DROP this pattern matching is infact quite effective and noise-free, resulting in the annotations acting as strong supervision.

While NMN can only handle the 6 reasoning categories that the supervision was tailored to, WNSMN focuses on the full DROP with numerical answers (called DROP-*num*) that involves more diverse reasoning on more open-ended questions. We empirically compare WNSMN on DROP-*num* with NMN and GenBERT that allow learning with partial or no strong supervision. Our results showcase that the proposed WNSMN achieves 32% better accuracy than NMN in absence of at least one or more types of supervision and performs 8% better than GenBERT when the latter is fine-tuned only on DROP in a comparable setup, without additional synthetic data having explicit supervision. We do not consider the SoTA Reasoning-free hybrid models as our baselines, as unlike NMN and GenBERT, they do not have any provision to learn in absence of the additional supervision generated through exhaustive enumeration and manual inspection.

Model

Our proposed WNSMN learns numerical-reasoning with sole weak supervision of the answer, by generating noisy heuristic program from the query parse structure and learning to execute it through explicit reasoning.

Parsing Query into Programs

To keep the framework generic, we use a simplified representation of the dependency parse (Chen and Manning 2014) of the query to get a generalized program, which is oblivious to the reasoning type. As shown in Figure 2, we first construct a node for the subtree rooted at each immediate child of the root by merging its descendants in the original word order. Next an edge is added from the left-most node (which we call the *root clause*) to every other node. Then by traversing left to right, each of these nodes are organized into program steps having a linear flow. For instance, the program obtained in Figure 1 is: *X1 = ('which is the longest')*; *X2 = ('goal by Carpenter', X1)*; *Answer = Discrete-Reasoning('which is the longest', X2)*.

Each program step consists of two types of arguments: (*i*) Query Span Argument obtained from the corresponding node indicating the query segment referred to in that step (*e.g., 'goal by Carpenter'* in Step 2), and (ii) Reference Argument(s) obtained from the incoming edges to that node from past steps, referring to the previous steps of the program that the current step depends on (*e.g., X1* in Step 2). Next, a final step of the program is added, which has the reference argument as the leaf node(s) obtained in the above manner and the query span argument as the root-clause. This step is specifically responsible for handling the discrete operation, enabled by the root-clause which is often indicative of the kind of discrete reasoning involved (*e.g., max*). However, this being a noisy heuristic, the QA model needs to be robust to such noise and additionally rely on the full query representation in order to predict the discrete operation type. For simplicity, we limit the reference arguments to 2.

Program Execution

To execute the program over the passage WNSMN first does **Entity Extraction** i.e. identifying the relevant entities from the passage and query, and maintaining them as separate canonicalized entity-lists along with their mention locations. For DROP, the only relevant entity types which involve numerical reasoning are numbers and dates. However, our model can also support other entity types that can be symbolically defined and whose mentions can be parsed and extracted from the context in a similar preprocessing step. Next, WNSMN learns an **Entity-Specific Cross-Attention** to model the passage information relevant to each step of the decomposed query and learn an attention distribution over the passage entities *w.r.t.* their query-relevance. Finally, it models **Explicit Discrete Reasoning** over the entities, by employing neural modules that *learn to sample* the operation and the relevant entity arguments and symbolic modules to execute the actual operation. It is then trained end-to-end in a **Reinforcement Learning (RL) Framework** with the answer as the sole supervision.

Entity-Specific Cross Attention Starting with the noisy program form of the query, a key requirement is to learn

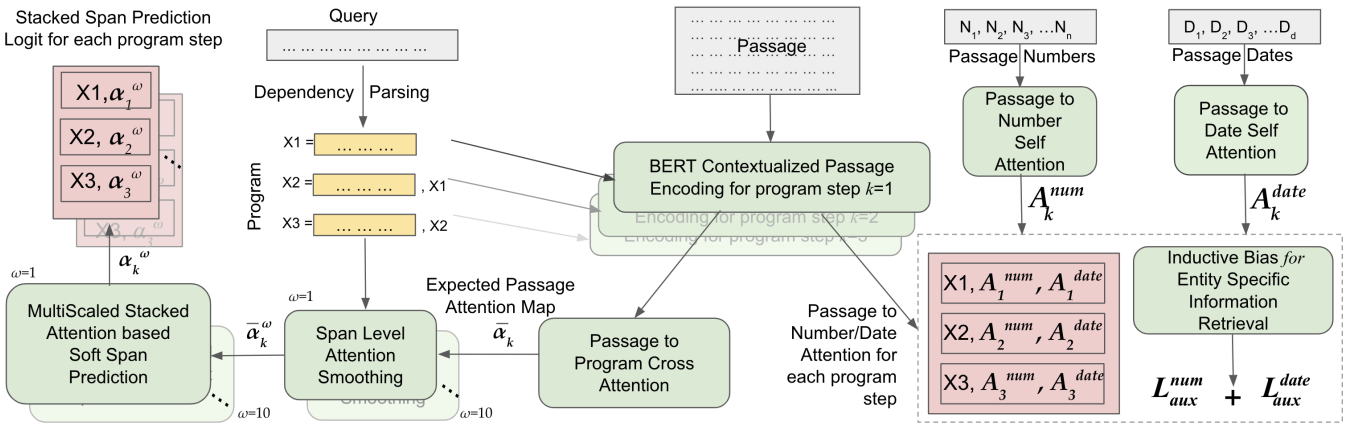


Figure 3: Modeling the interactions between the passage and program (left), and between the passage and its entities (right) (e.g. in this figure we take *number* and *date* as entities which are prevalent in DROP dataset). For a program step, they respectively yield Stacked Span Prediction Logits and Attention over the entities for each passage token (in pink boxes). These are linearly combined to get the expected distribution over the entities.

the attention map over the entities of each type extracted from the passage, for each program step. This is achieved by a BERT based model, we describe below, which learns the attention between each program step and the passage and the attention between the passage text and passage entities. The other SoTA models like NMN, NeRd also have a BERT-like reader model to read query related information from the passage and a programmer model to decode the specialized programs. But these components and their training becomes significantly simplified due of the availability of heuristic or handcrafted programs supervision.

Program-Passage Interaction This module (Figure 3, left) learns to associate *query span* arguments of the program with the passage. For this, similar to NMN, we use a BERT-base pretrained encoder (Devlin et al. 2018) to get contextualized token embeddings of the passage and query span argument of each program step, respectively denoted by $\boldsymbol{P}_k$ and $\boldsymbol{Q}_k$ for the k'th program step. Based on it, we learn a *similarity* matrix $\mathbf{S} \in \mathbb{R}^{l \times n \times m}$ between the program and passage, where l, n, and m respectively are the program length, query span length and passage length (in tokens). Each $\boldsymbol{S}_k \in \mathbb{R}^{n \times m}$ represents the affinity over the passage tokens for the k'th program argument and is defined as $\boldsymbol{S}_k(i,j) = \boldsymbol{w}^T[\boldsymbol{Q}_{ki}; \boldsymbol{P}_{kj}; \boldsymbol{Q}_{ki} \odot \boldsymbol{P}_{kj}]$, where $\boldsymbol{w}$ is a learnable parameter vector and $\odot$ denotes element-wise multiplication. From this, an attention map $\boldsymbol{A}_k$ is computed over the passage tokens for the k'th program argument as $\boldsymbol{A}_k(i,j) = \text{softmax}_j(\boldsymbol{S}_k(i,j)) = \frac{\exp(\boldsymbol{S}_k(i,j))}{\sum_j \exp(\boldsymbol{S}_k(i,j))}$. Similarly, for the i'th token of the k'th program argument the cumulative attention a_{ki} *w.r.t.* the passage is $a_{ki} = \text{softmax}_i(\sum_j \boldsymbol{S}_k(i,j))$. A linear combination of the attention map $\boldsymbol{A}_k(i,\cdot)$ weighted by a_{ki} gives the expected passage attention for the k'th program step, $\bar{\boldsymbol{\alpha}}_k = \sum_i a_{ki} \boldsymbol{A}_k(i,\cdot) \in \mathbb{R}^m$.

Further, to facilitate information spotting and extraction over contiguous spans of text, we regularize the passage attention through heuristic span-level smoothing technique (Huang et al. 2020), by averaging the token-level attention over different sliding windows of lengths $\omega = \{1, 2, \ldots, 10\}$, to spread the attention at different levels of granularity. We use the same scale-parameters as used in the primary baseline work NMN. This results in 10 different attention maps over the passage for the k'th step of the program: $\{\bar{\boldsymbol{\alpha}}_k^\omega | \omega \in \{1, 2, \ldots, 10\}\}$. Next, a multi-scaled version of $\bar{\boldsymbol{\alpha}}_k^\omega$ is obtained by multiplied with scaling factors $\boldsymbol{s} = \{1, 2, 5, 10\}$ to yield a $|\boldsymbol{s}|$-dimensional representation for each passage token. It is then passed through a L-layered stacked self-attention *transformer* block (Vaswani et al. 2017) followed by a *linear layer* to obtain the span prediction logits $\boldsymbol{\alpha}_k^\omega$, for each window length ω. Further, the span prediction logits at each program step (say k) is additively combined with those from the previous steps referenced in the current one, through the reference argument ($ref(k)$) at step k, *i.e.,* $\boldsymbol{\alpha}_k^\omega = \boldsymbol{\alpha}_k^\omega + \sum_{k' \in ref(k)} \boldsymbol{\alpha}_{k'}^\omega$.

Passage-Entity Interaction This module (Figure 3, right) facilitates an entity-based information spotting capability, that is, given a passage mention of an entity (number/date type) relevant to the query, the model should be able to attend to the neighborhood around it. To do this, for each entity type t, we first compute *passage to entity tokens* attention maps $\mathbf{A}^t \in \mathbb{R}^{l \times m \times N}$ with N being the number of unique entities of type t. Note that these attention maps are different for each program step k as the contextual BERT encoding of the passage tokens ($\boldsymbol{P}_k$) is coupled with the program's span argument of that step. Specifically, at the k-th step, $\boldsymbol{A}_k^t(i,\cdot)$ denotes the probability distribution over the N unique entity tokens of type t *w.r.t.* the i-th passage token. The attention maps are obtained by a softmax normalization of each row of the corresponding *passage to entity tokens* similarity matrix, $\boldsymbol{S}_k^t \in \mathbb{R}^{m \times N}$ for $k = \{1, \ldots, l\}$, where the elements of $\boldsymbol{S}_k^t$ are computed as $\boldsymbol{S}_k^t(i,j) = \boldsymbol{P}_{ki}^T \boldsymbol{W}_t \boldsymbol{P}_{kn_j}$ with $\boldsymbol{W}_t \in \mathbb{R}^{d \times d}$ being a learnable projection matrix for type t and n_j being the passage location of the j-th entity token. The similarity scores are additively aggregated over

all passage mentions of the same entity.

Program-Entity Interaction The relation between program and entities is then modeled by combining the *passage to entity* attention map A^t and the per-step passage span logits $\boldsymbol{\alpha}_k^\omega$ to get $\boldsymbol{\tau}_k^\omega = \text{softmax}(\sum_i \alpha_{ki}^\omega \boldsymbol{A}_k^t(i,\cdot)) \in \mathbb{R}^N$. This gives the expected distribution over the N entity tokens of type t for the k-th program step and using ω as the smoothing window size. The final stacked attention maps obtained for the different windows are $\mathcal{T}_k^t = \{\boldsymbol{\tau}_k^\omega | \omega \in \{1, 2, \ldots 10\}\}$.

A critical requirement for a meaningful attention over entities of type t, is to incorporate information extraction capability in the attention maps $\mathbf{A}^t$, by enabling the model to attend over the neighborhood of the relevant entity mentions. This is achieved by minimizing the unsupervised auxiliary loss $\mathcal{L}_{aux}^t$ in the training objective, which impose an inductive bias over the number and date entities, similar to (Gupta et al. 2020). Its purpose is to ensure that the passage attention is densely distributed inside the neighborhood of $\pm\ \Omega$ (*e.g.,* 10) of the passage location of the entity mention, without imposing any bias on the attention distribution outside the neighborhood. Consequently, it maximises the log-form of cumulative likelihood of the attention distribution inside the window and the entropy of the attention distribution outside it.

$$\mathcal{L}_{aux}^t = -\frac{1}{l}\sum_{k=1}^{l}\Bigg[\sum_{i=1}^{m}[\log(\sum_{j=1}^{N}\mathbb{1}_{n_j \in [i \pm\ \Omega]}\boldsymbol{A}_k^t(i,j)) - \sum_{j=1}^{N}\mathbb{1}_{n_j \notin [i \pm\ \Omega]}\boldsymbol{A}_k^t(i,j)\log(\boldsymbol{A}_k^t(i,j))]\Bigg]$$

Modeling Discrete Reasoning The model next learns to execute a single step[1] of discrete reasoning based on the final program step which contains the (*i*) root-clause of the query which often indicates the discrete operation type (*e.g., 'how many goals'* indicates `count`), and (*ii*) reference argument indicating the previous program steps the final step depends on. For entity type t (date/number), each previous step (say k) is represented as stacked attention maps $\mathcal{T}_k^t$. $\mathcal{T}^t$ from the previous program steps is one of the main inputs to the Argument Sampling transformer network in the Discrete Reasoning module. The Discrete Reasoning takes the last step of the program e.g. (for *DiscreteReasoning('Which is the longest', X2)*, the second input to the network would be $\mathcal{T}_2^t$). Without learning the interaction between the Program, Passage and Entities, this input to the Argument Sampler will not be meaningful. With this input, this module learns to select the correct discrete operation (here *max* to find the longest) on the attention-map $\mathcal{T}_2^t$ learnt over the entities of type t in the previous step.

Operator Sampling Network The operator network takes as input, (*i*) BERT's `[CLS]` representation for the passage-query pair and LSTM encoding (randomly initialized) of the BERT contextual representation (*w.r.t.* the passage) of (*ii*) root-clause from the final program step and (*iii*) full query. The BERT contextualized representation allows the model to use the key passage information during operator sampling and having access to the full query allows it to learn even when the parsing is noisy. With this input, it predicts the following two entity and operator:

- *Entity-Type Predictor Network*, an Exponential Linear Unit (Elu) activated fully-connected layer followed by a softmax that outputs the probabilities of sampling either date or number types.
- *Operator Predictor Network*, an Elu-activated fully connected layer followed by a softmax to learn probability distribution over a fixed catalog of 6 numerical and logical operations (`count`, `max`, `min`, `sum`, `diff`, `negate`), each represented with learnable embeddings.

Apart from `diff` which acts only on two arguments, all other operations can take arbitrarily many arguments.

Argument Sampling Network This network learns to sample entities of type t (*e.g.,* date/numbers) as arguments for the sampled discrete operation given the entity-specific stacked attentions $\mathcal{T}_k^t$ for each previous step k that appears as the reference argument in the final program step. To allow sampling of fixed or arbitrary number of arguments, the argument sampler learns four types of networks, each modeled with an L-layered self attention based *transformer* block with an output dimension of d followed by different non-linear layers embodying their functionality and a softmax normalization to get the sampling probability.

- *Sample $n \in \{1, 2\}$ Argument Module:* Outputs a distribution over the single entities (n = 1) or a joint distribution over the entity-pairs (n = 2); defined as $\text{softmax}(Elu(Linear_{d\times n}(Transformer(\mathcal{T}))))$.
- *Counter Module:* Predicts a distribution over number ($\leq$10) of entity arguments to sample through $\text{softmax}(Elu(Linear_{d\times 10}(CNN\ (Transformer(\mathcal{T})))))$.
- *Entity-Ranker Module:* Learns to rerank the entities and outputs a distribution over all the entities given the stacked attention maps as input. $\text{softmax}(PRelu(Linear_{d\times 1}(Transformer(\mathcal{T}))))$
- *Sample Arbitrary Argument:* Learns to sample n (n being prediction of Counter Module) entities from a multinomial over the entity distribution predicted by the Entity Ranker.

Depending on the number of arguments needed by the discrete operation and the number of reference arguments in the final program step, the model invokes one of *Sample {1, 2, Arbitrary} Argument*. For instance, if the sampled operator is `diff` which needs 2 arguments, and the final step has 1 or 2 reference arguments, then the model respectively invokes either *Sample 2 argument* or *Sample 1 argument* on the stacked attention $\mathcal{T}$ corresponding to each reference argument. For the *Arbitrary Argument* case, the model first predicts the number of entities (c) to sample using the Counter Network, and then samples from the multinomial distribution over the joint of c-combinations of entities constructed from the output distribution of the Entity Ranker.

[1] A reasonable assumption for DROP with a recall of 90% on the training set. However, it does not limit the generalizability of WNSMN, as with the standard beam search it is possible to scale to an l-step Markov Decision Process (MDP).

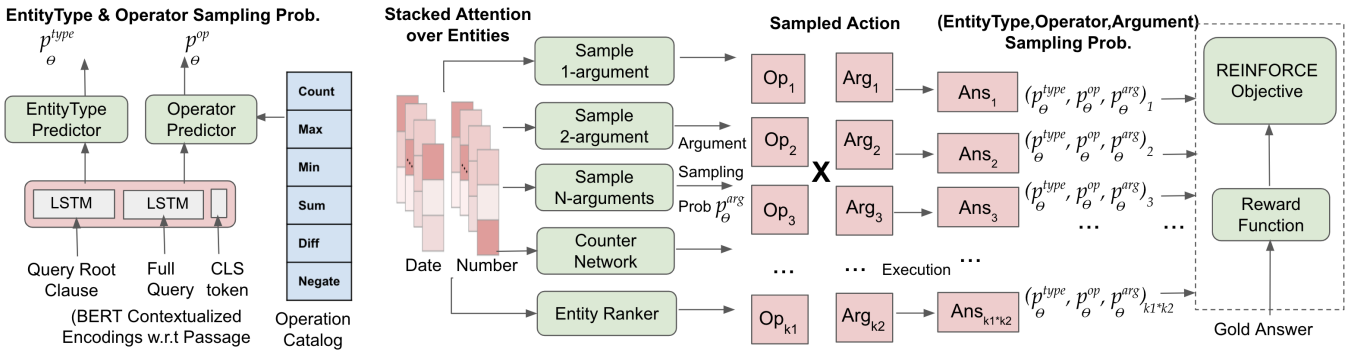


Figure 4: Operator and Argument Sampling Networks and RL framework over sampled discrete actions.

RL Training with Weak Supervision We use an RL framework to train the model with only discrete binary feedback from the exact match of the gold and predicted numerical answer. In particular, we use the REINFORCE (Williams 1992) policy gradient method where a stochastic policy comprising a sequence of actions is learned with the goal of maximizing the expected reward. In our case, the discrete operation along with argument sampling constitute the *action*. However, because of our assumption that a single step of discrete reasoning suffices for most questions in DROP, we further simplify the RL framework to a contextual multi-arm bandit (MAB) problem with a 1-step MDP, *i.e.,* the agent performs only one step action. Despite the simplifying assumption of 1-step MDP, the following characteristics of the problem render it highly challenging: (*i*) the action space $\mathcal{A}$ is exponential in the order of number of operations and argument entities in the passage (averaging to *12K* actions for DROP-*num*); (*ii*) the extreme reward sparsity owing to the binary feedback is further exacerbated by the presence of spurious rewards, as the same answer can be generated by multiple diverse actions. Previous approaches like NMN can avoid such spurious supervision because of the heuristically obtained additional annotation of the question category, the gold program or its execution atleast for some training instances.

In our contextual MAB framework, for an input x = (passage(p), query(q)), the context or environment state $s_\phi(x)$ is modeled by the entity specific cross attention, parameterized by ϕ) between the (*i*) passage (*ii*) program-form of the query and (*iii*) extracted passage date/number entities. Given the state $s_\phi(x)$, the layout policy (parameterized by θ) then learns the query-specific inference layout, *i.e.,* the discrete action sampling policy $P_\theta(a|s_\phi(x))$ for action $a \in \mathcal{A}$. The action sampling probability is a product of the probability of sampling entities from the appropriate entity type (P_θ^{type}), the operator (P_θ^{op}), and the entity argument(s) (P_θ^{arg}), normalized by number of arguments to sample. Therefore, with the learnable context representation $s_\phi(x)$ of input x, the end-to-end objective is to jointly learn $\{\theta, \phi\}$ that maximises the expected reward $R(x, a) \in \{-1, +1\}$ over the sampled actions (a), based on the answer match. To mitigate the learning instability in such sparse confounding reward settings, we intialize with a simpler iterative *hard-Expectation Maximization (EM)* learning objective, called Iterative Maximal Likelihood (IML) (Liang et al. 2017). With the assumption that the sampled actions are extensive enough to contain the gold answer, IML greedily searches for the *good* actions by fixing the policy parameters, and then maximises the likelihood of the *best* action that led to the highest reward. We define *good* actions ($\mathcal{A}^{good}$) as those that result in the gold answer itself and take a conservative approach of defining *best* among them as simply the most likely one according to the current policy.

$$J^{IML}(\theta, \phi) = \sum_x \max_{a \in \mathcal{A}^{good}} \log P_{\theta,\phi}(a|x)$$

After the IML initialization, we switch to REINFORCE as the learning objective where the goal is to maximise the expected reward ($J^{RL}(\theta, \phi) = \sum_x \mathbb{E}_{P_{\theta,\phi}(a|x)} R(x, a)$) as

$$\nabla_{(\theta,\phi)} J^{RL} = \sum_x \sum_{a \in \mathcal{A}} P_{\theta,\phi}(a|x)(R(x, a) - B(x))\nabla_{\theta,\phi}(\log P_{\theta,\phi}(a|x))$$

$B(x)$ being average (baseline) reward for instance x. Further, in order to mitigate overfitting, in addition to L_2-regularization and dropout, we also add entropy based regularization over the argument sampling distribution, in each of the sampling networks. Other than the iterative ML initialization, the program entity interaction also provides an self-supervised auxiliary loss ($\mathcal{L}^t_{aux}$ for entity type t). This inductive bias significantly helps in stabilizing the training.

Experiments

We now empirically compare the *exact-match* performance of WNSMN with SoTA baselines on versions of DROP dataset and also examine how it fares in comparison to strong supervised skylines. The **Primary Baselines** for WNSMN are the explicit reasoning based **NMN** (Gupta et al. 2020) which uses additional strong supervision and the BERT based language model **GenBERT** (Geva, Gupta, and Berant 2020) that does not embody any reasoning and autoregressively generates numeric answer tokens. As the **Primary Dataset** we use **DROP-*num***, the subset of DROP with numerical answers. This subset contains 45K and 5.8K instances respectively from the standard DROP train and development sets. Originally, NMN was showcased on a very

specific subset of DROP, restricted to the 6 reasoning-types it could handle, out of which three (*count*, *date-difference*, *extract-number*) have numeric answers. This subset comprises 20K training and 1.8K development instances, out of which only 10K and 800 instances respectively have numerical answers. We further evaluate on this numerical subset, referred to as **DROP-Pruned-*num***.In both cases, training data was randomly split into 70%:30% for train and validation and the standard DROP development set was treated as Test.

Figure 5 shows the t-SNE plot of pretrained Sentence-BERT (Reimers and Gurevych 2019) encoding of *all* questions in DROP-*num*-Test and also the DROP-Pruned-*num*-Test subset with different colors (red, green, yellow) representing different types. Not only are the DROP-*num* questions more diverse than the carefully chosen DROP-Pruned-*num* subset, the latter also forms well-separated clusters corresponding to the three reasoning types. Additionally, the average perplexity of the DROP-Pruned-*num* and DROP-*num* questions was found to be *3.9* and *10.65* respectively, further indicating the comparatively open-ended nature of the former.

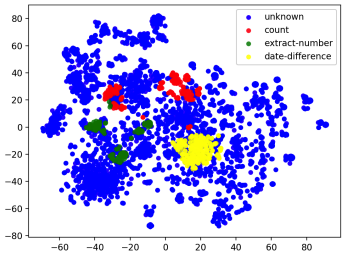


Figure 5: t-SNE of DROP-*num*-Test questions.

For the primary baselines NMN and GenBERT, we report the performance on in-house trained models on the respective datasets, using the code open-sourced by the authors. The remaining results, taken from (Geva, Gupta, and Berant 2020), (Kinley and Lin 2019), and (Ran et al. 2019); refer to models trained on the full DROP dataset. All models use the same pretrained BERT-base. Also note that a primary requirement of all models other than GenBERT and WNSMN *i.e.,* for NMN, MTMSN, NABERT, NAQANET, NumNet, is the exhaustive enumeration of the output space of all possible discrete operations. This simplifies the QA task to a classification setting, thus alleviating the need for discrete reasoning in the inference processs.

Supervision Type Prog.	Exec.	QAtt.	Acc. (%)
NMN-*num* variants			
✗	✓	✓	11.77
✓	✗	✓	17.52
✓	✓	✗	18.27
✓	✗	✗	18.54
✗	✓	✗	12.27
✗	✗	✓	11.80
✗	✗	✗	11.70
GenBERT			
✗	✗	✗	42.30
GenBERT-*num*			
✗	✗	✗	38.41
WNSMN			
✗	✗	✗	**50.97**

Table 1: Baselines and WNSMN on DROP-*num*-Test.

Table 1 presents our primary results on **DROP-*num***, comparing the performance of WNSMN (accuracy of the top-1 action sampled by the RL agent) with various ablations of NMN (provided in the authors' implementation) by removing atleast one of **Prog**ram, **Exec**ution, and **Q**uery **Att**ention supervision and GenBERT models with pretrained BERT that are finetuned on DROP or DROP-*num* (denoted as GenBERT and GenBERT-*num*).

For a fair comparison with our weakly supervised model, we do not treat NMN with all forms of supervision or GenBERT pretrained with additional *synthetic* numerical and textual data as comparable baselines. These GenBERT variants indeed enjoy strong reasoning supervision in terms of gold arithmetic expressions in these auxiliary datasets. Also, the synthetic pretraining dataset are not generic enough and are quite tailored to the target dataset, enjoying strikingly similarity to DROP. This is reflected in the domains (*nfl* and *history*), passage vocabulary and distribution of the range of numerical entities and the language and nature of questions.

However, NMN's performance, in this setting, is abysmally poor, indeed a drastic degradation in comparison to its performance on the pruned DROP subset reported by (Gupta et al. 2020) and in our subsequent experiments in Table 2. This can be attributed to their limitation in handling more diverse classes of reasoning and open-ended queries in DROP-*num*, further exacerbated by the lack of one or more types of strong supervision.[2] Our earlier analysis on the complexity of the questions in the subset and full DROP-*num* further quantify the relative difficulty level of the latter. On the other hand, GenBERT delivers a mediocre performance, while GenBERT-*num* degrades additionally by 4%, as learning from numerical answers alone further curbs the language modeling ability. Our model performs significantly better than both these baselines, surpassing GenBERT by 8% and the NMN baseline by around 32%. This showcases the significance of incorporating explicit reasoning in neural models in comparison to the vanila large scale LMs like GenBERT. It also establishes the generalizability of such reasoning based models to more open-ended forms of QA, in comparison to contemporary modular networks like NMN, owing to its ability to handle both learnable and discrete modules in an end-to-end manner.

Next, in Table 2, we compare the performance of the proposed WNSMN with the same NMN variants (as in Table 1) on **DROP-Pruned-*num***. Some of the salient observations are: (*i*) WNSMN in fact reaches a performance quite close to the *strongly supervised*

Supervision-Type Prog.	Exec.	QAtt.	Acc.	Count	Extract-num	Date-differ
NMN-*num* models						
✓	✓	✓	**68.6**	50.0	**88.4**	72.5
✗	✓	✓	42.4	24.1	73.9	36.4
✓	✗	✓	54.3	47.9	80.7	40.9
✓	✓	✗	63.3	45.5	81.1	68.7
✗	✗	✓	48.2	38.1	72.4	41.9
✓	✗	✗	61.0	44.7	81.1	63.2
✗	✓	✗	62.3	43.7	84.1	67.7
✗	✗	✗	62.1	46.8	83.6	66.1
WNSMN						
✗	✗	✗	66.5	**58.8**	66.8	**75.2**

Table 2: NMN-*num* and WNSMN on DROP-Pruned-*num*-Test

.

[2]Both the results and limitations of NMN in Table1 and 2 were confirmed by the authors of NMN as well.

NMN variant (first row), and is able to attain at least an improvement margin of 4% over all other variants obtained by removing one or more types of supervision. This is despite all variants of NMN *additionally* enjoying the exhaustive precomputation of the output space of possible numerical answers; (*ii*) WNSMN suffers only in the case of *extract-number* type operations (*e.g., max,min*) that involve a more complex process of sampling arbitrary number of arguments (*iii*) Performance drop of NMN is not very large when all or none of the strong supervision is present, possibly because of the limited diversity over reasoning types and query language; and (*iv*) Query-Attention supervision infact adversely affects NMN's performance, in absence of the *program* and *execution* supervision or both, possibly owing to an undesirable biasing effect. However when both supervisions are available, query-attention is able to improve the model performance by 5%. Further, we believe the test set of 800 instances is too small to get an unbiased reflection of the model's performances.

In Table 3, we additionally inspect recall over the top-k actions sampled by WNSMN to estimate how it fares in comparison to the strongly supervised skylines: (*i*) NMN with all forms of strong supervision; (*ii*) GenBERT variants +ND, +TD and +ND+TD further pretrained on synthetic **N**umerical and **T**extual **D**ata and both; (*iii*) reasoning-free hybrid models like MTMSN (Hu et al. 2019) and NumNet (Ran et al. 2019), NAQANet (Dua et al. 2019) and NABERT, NABERT+ (Kinley and Lin 2019). Note that both NumNet and NAQANet do not use pretrained BERT. Out of the strong supervised models, MTMSN achieves SoTA performance through a supervised framework of training specialized predictors for each reasoning type to predict the numerical expression directly instead of learning to reason. While top-1 performance of WNSMN (in Table 1) is 4% worser than NABERT, Recall@top-2 is equivalent to the strongly supervised NMN, top-5 and top-10 is comparable to NABERT+, NumNet and GenBERT models +ND, +TD and top-20 nearly achieves SoTA. This promising recall suggests that more sophisticated RL algorithms with better exploration strategies can possibly bridge this performance gap.

Strong Supervised Models	**Acc. (%)**
NMN-*num* (all supervision)	58.10
GenBERT+ND	69.20
GenBERT+TD	70.50
GenBERT+ND+TD	75.20
NAQANet	44.97
NABERT	54.27
NABERT+	66.60
NumNet	69.74
MTMSN	75.00

Recall@top-k actions of WNSMN(%)					
$k=2$	$k=3$	$k=4$	$k=5$	$k=10$	$k=20$
58.6	63.0	65.4	67.4	72.3	74.2

Table 3: Skylines & WNSMN top-k performance on DROP-*num*-Test

Analysis

In this section, we perform more analysis of the proposed model and motivate some future directions.

Performance Analysis Despite the notorious instabilities of RL due to high variance, the training trend, as shown in Figure 5(a) is not afflicted by catastrophic forgetting. The sudden performance jump between epochs 10-15 is because of switching from iterative ML initialization to REINFORCE objective. Figure 5(b) shows the individual module-wise performance evaluated using the noisy pseudo-rewards, that indicate whether the action sampled by this module *led* to the correct answer or not. Accordingly, we define pseudo-reward for sampling an operator as the maximum of the reward obtained from all the actions involving that operator. Next in Figure 5(c), we bucket the performance of the baselines and WNSMN by the total number of passage entities. From there we observe that WNSMN remains unaffected by the increasing number of entities, despite the action space explosion. On the other hand, GenBERT's performance drops linearly beyond 25 entities and NMN-*num* degrades exponentially from beginning, owing to its dependency on the exponentially growing exhaustively precomputed output space. Further in Fig 6, we bucket the performance of WNSMN by the query-length and observe that our performance degrades only slightly for DROP-num-Test and DROP-Pruned-num-Test with increasing query length.

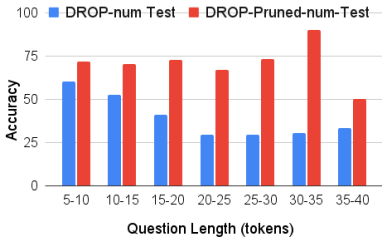
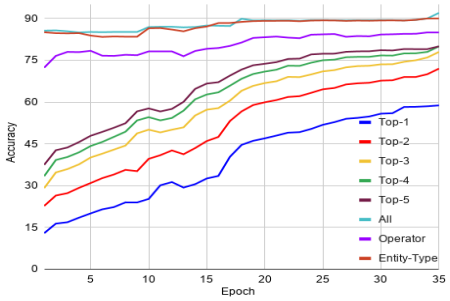


Figure 6: Performance of WNSMN by varying Question Length

Manual Analysis Since WNSMN learns to reason in an entirely weak supervised framework, in Table 4 we manually inspected 340 instances from DROP-num Test where the gold answer is reachable through different operations, to understand in how many cases the top-1 action predicted by model is indeed correct or when it is spurious. The notable conclusion from this is that only in 28 out of the manually annotated 340 instances, the the model's reasoning is spuriously correct. However, out of the cases where atleast one of the top-50 actions is a *good* we observe that the model is able to learn when the answer is directly present as an entity or can be obtained by operating other entities and when it needs to count over multiple entity

Good Action: i.e. Resulting in True Answer	
Correct Action: i.e. Manually Annotated Correct	
A:- Instances with atleast 1 good action	4868
B: Instances with multiple good action	2533
C:- Instances with top-1 action good	2956
D:- Intersection of set **B** and **C**	620
E:- Instances manually annotated from **D**	340
F:- Instances from **E** with Correct top-1	312
Num. of good actions per instance in **A**	1.5
Num. of good actions per instance in **B**	2.25

Table 4: Manual Analysis of WNSMN predictions on 5.8K DROP-*num* Test Instances.



Module	Performance
Sample 1 Argument	54% (Accuracy)
Sample 2 Argument	52% —"—
Counter	50% —"—
Entity Ranker	53% —"—
Operator Predictor	78% —"—
Entity Type Predictor	83% —"—
Overall Action Sampler	84% (Recall@All)

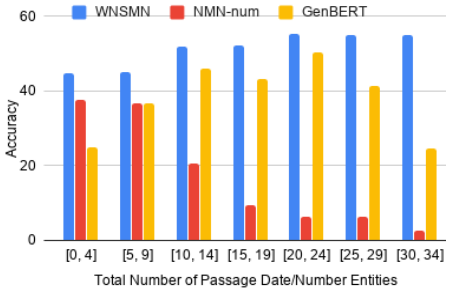


Figure 5: (a) Training trend showing the Recall@top-k actions and accuracy of Operator and Entity-type Predictor, estimated based on noisy psuedo rewards, (b) Module-wise performance (using pseudo-reward) on DROP-*num*-Test, (c) Bucketing performance by total number of passage (date and number) entities for WNSMN, and the best performing baseline NMN and GenBERT model taken from Table 1.

mentions.

More Stable RL Framework The training trend in Figure 5(a) shows early saturation and the module-wise performance indicates overfitting despite some regularization tricks. While more stable RL algorithms like Actor-Critic, Trust Region Policy Optimization (Schulman et al. 2015) or Memory Augmented Policy Optimization (Liang et al. 2018) can mitigate these issues, we leave them for future exploration. Also, though this work's objective was to train module networks with weak supervision, the sparse confounding rewards in the exponential action space indeed render the RL training quite challenging. One practical future direction to bridge the performance gap would be to pretrain with strong supervision on at least a subset of reasoning categories or on more constrained forms of synthetic questions, similar to GenBERT. This would require inspection and evaluation of generalizability of the RL model to unknown reasoning types or more open-ended questions.

Conclusion

In this work, we presented Weakly Supervised Neuro-Symbolic Module Network for numerical reasoning based MRC based on a generalized framework of query parsing to noisy heuristic programs. It trains both neural and discrete reasoning modules end-to-end in a Deep Reinforcement Learning framework with only discrete reward based on exact answer match. Our empirical analysis on the *numerical-answer only subset* of DROP showcases significant performance improvement of the proposed model over SoTA Neural Module Network and Transformer based language model GenBERT, when trained in comparable weakly supervised settings. While, to our knowledge, this is the first effort towards training modular networks for fuzzy reasoning over RC in a weakly-supervised setting, there is significant scope of improvement, such as employing more sophisticated reinforcement learning framework or by leveraging the pretraining of explicit reasoning.

References

Chen, D.; and Manning, C. 2014. A Fast and Accurate Dependency Parser using Neural Networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 740–750. Doha, Qatar: Association for Computational Linguistics.

Chen, X.; Liang, C.; Yu, A. W.; Zhou, D.; Song, D.; and Le, Q. V. 2020. Neural Symbolic Reader: Scalable Integration of Distributed and Symbolic Representations for Reading Comprehension. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Cite arxiv:1810.04805Comment: 13 pages.

Dua, D.; Wang, Y.; Dasigi, P.; Stanovsky, G.; Singh, S.; and Gardner, M. 2019. DROP: A Reading Comprehension Benchmark Requiring Discrete Reasoning Over Paragraphs. In *Proc. of NAACL*.

Geva, M.; Gupta, A.; and Berant, J. 2020. Injecting Numerical Reasoning Skills into Language Models. In *ACL*.

Gupta, N.; Lin, K.; Roth, D.; Singh, S.; and Gardner, M. 2020. Neural Module Networks for Reasoning over Text. In *International Conference on Learning Representations*.

Hu, M.; Peng, Y.; Huang, Z.; and Li, D. 2019. A Multi-Type Multi-Span Network for Reading Comprehension that Requires Discrete Reasoning. In *Proceedings of EMNLP*.

Huang, T.; Deng, Z.; Shen, G.; and Chen, X. 2020. A Window-Based Self-Attention approach for sentence encoding. *Neurocomputing*, 375: 25–31.

Kinley, J.; and Lin, R. 2019. NABERT+: Improving Numerical Reasoning in Reading Comprehension.

Koncel-Kedziorski, R.; Roy, S.; Amini, A.; Kushman, N.; and Hajishirzi, H. 2016. MAWPS: A Math Word Problem Repository. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1152–1157. San Diego, California: Association for Computational Linguistics.

Liang, C.; Berant, J.; Le, Q.; Forbus, K. D.; and Lao, N. 2017. Neural Symbolic Machines: Learning Semantic

Parsers on Freebase with Weak Supervision. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, 23–33.

Liang, C.; Norouzi, M.; Berant, J.; Le, Q. V.; and Lao, N. 2018. Memory Augmented Policy Optimization for Program Synthesis and Semantic Parsing. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 31*, 10015–10027. Curran Associates, Inc.

Neelakantan, A.; Le, Q. V.; Abadi, M.; McCallum, A.; and Amodei, D. 2017. Learning a Natural Language Interface with Neural Programmer. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Ran, Q.; Lin, Y.; Li, P.; Zhou, J.; and Liu, Z. 2019. NumNet: Machine Reading Comprehension with Numerical Reasoning. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2474–2484. Hong Kong, China: Association for Computational Linguistics.

Reimers, N.; and Gurevych, I. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Roy, S.; and Roth, D. 2015. Solving General Arithmetic Word Problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 1743–1752. Lisbon, Portugal: Association for Computational Linguistics.

Saha, A.; Ansari, G. A.; Laddha, A.; Sankaranarayanan, K.; and Chakrabarti, S. 2019. Complex Program Induction for Querying Knowledge Bases in the Absence of Gold Programs. *Transactions of the Association for Computational Linguistics*, 7: 185–200.

Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; and Moritz, P. 2015. Trust Region Policy Optimization. volume 37 of *Proceedings of Machine Learning Research*, 1889–1897. Lille, France: PMLR.

Subramanian, S.; Bogin, B.; Gupta, N.; Wolfson, T.; Singh, S.; Berant, J.; and Gardner, M. 2020. Obtaining Faithful Interpretations from Compositional Neural Networks. In *Association for Computational Linguistics (ACL)*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L. u.; and Polosukhin, I. 2017. Attention is All you Need. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 30*, 5998–6008. Curran Associates, Inc.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8: 229–256.

Zhong, V.; Xiong, C.; and Socher, R. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. arXiv:1709.00103.