UPDATER-EXTRACTOR ARCHITECTURE FOR INDUCTIVE WORLD STATE REPRESENTATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Developing sequential models traditionally involves two stages - training and application. Retention of information acquired after training (at application time) is architecturally limited by the size of the model's context window (in the case of transformers), or by the practical difficulties associated with long sequences (in the case of RNNs). In this paper, we propose a novel one-step training procedure that makes such training feasible. First, we provide the theoretical justification for this training procedure. Second, to demonstrate how our result can be applied in practice, we propose a transformer-based Updater-Extractor architecture that can work with sequences of arbitrary length and refine its long-term knowledge about the world based on inputs at application time. Empirically, we investigate the model performance on a variety of different tasks: we use two new simulated tasks tasks to study the model's ability to handle extremely long-range dependencies, then we demonstrate competitive performance on the challenging Pathfinder problem using vanilla attention.

1 INTRODUCTION

Recent years have been marked by numerous successes of sequential models. The development of the transformer architecture has had an especially broad impact on a range of disciplines, from Natural Language Processing (Vaswani et al., 2017; Radford et al., 2019) to Computer Vision (Khan et al., 2021). Despite their successes, the absence of persistent world state representations in transformer architectures limits information retention horizon to the length of the context window, making it challenging to work with extremely long sequences.

Due to the large size of transformer models, adding a persistent world state representation becomes problematic: Backpropagation Through Time (Werbos, 1990) can not be used practically due to memory requirements, while Truncated Backpropagation Through Time (TBTT) lacks the theoretical guarantees and is known to be unstable.

This is especially daunting in the context of NLP applications: the absence of persistent world state representations means that transformers architecturally **lack the ability to learn through language**. At application time, there is no mechanism for **long-term changes in the model's knowledge or behaviour based on linguistic inputs**. Most models can process a few paragraphs of text and perform the task they were trained for, such as question answering, summarisation, named entity recognition, and so on. Unfortunately, however, they cannot retain knowledge between application-time instances. For example, summarising a Wikipedia article about World War II has no effect on the model's ability to summarise related articles in the future.

In contrast, when humans process natural language, they continuously *learn* from it. Reading a book or an article, as well as having a meaningful conversation with a friend may change our views on a wide variety of topics. These views, in turn, directly affect our personal and professional decisions. Thus, in humans, processing natural language is tightly bound to the problem of adjusting one's beliefs about the world, and can have a lasting impact on one's behaviour.

In our work, we propose an architecture modification and a training procedure that, taken together, resolve the problem. Architectural changes introduce persistent knowledge representations into transformers and allow us to repeatedly query the representation, while the new training proce-

dure allows us to obtain theoretical guarantees on using TBTT, making it possible to train the model on sequences of arbitrary lengths without propagating gradients through the world state.

1.1 MAIN CONTRIBUTIONS

- Identifying sufficient conditions on the data distribution under which cutting gradients between recurrent steps is justified both practically and theoretically.
- Proposing a novel transformer-based architecture with a consistent world state representation that can capitalize on the theoretical result above.
- Showing, on three simulated tasks, that the model has desirable world-state-tracking properties, strong generalization over episode length, and high tolerance to uncertainty and distribution shifts.
- Showing that the model achieves competitive performance on the Pathfinder problem, a challenging benchmark for long-range sequences, while reducing memory requirements by an order of magnitude.

2 RELATED WORK

Handling longer sequences with transformers There has been a recent push to create models that can handle extremely long sequences, including the development of attention mechanisms that can handle long sequences more efficiently (Choromanski et al., 2020; Wang et al., 2020). A number of benchmark tasks were proposed (Tay et al., 2020) to allow for systematic evaluation of such models. While these optimizations can be fruitful, they are far from the scale necessary to span lifetime-long experiences. Instead, we explore a recurrent approach that separates input processing and information internalization.

A number of models use an auxiliary data structure. Notable non-Transformer models include the Neural Turing Machine (Graves et al., 2014) and the Differentiable Neural Computer (Graves et al., 2016); however, training them on extremely long sequences is computationally prohibitive. Khandelwal et al. (2020) use efficient nearest neighbor lookups to retrieve information relevant to the task at hand. However, this is not suited for "on the fly" learning in a temporal context, as the lookup database is fixed at application time.

Transformer XL (Dai et al., 2019) re-uses activations from the previous segment to increase the temporal range of the model. Rae et al. (2019) extend this by additionally storing compressed memories that have been trained with an autoencoder loss in order to reconstruct old memories (activations of each layer at a previous time step). However, both models still have a finite temporal range and do not fully propagate gradients, and so cannot ensure that relevant information is kept, even within this range. Our work simplifies these architectures by maintaining a single state that contains all relevant information up to the present, with no range limit. Additionally, we provide a theoretical result that directly ensures that all relevant information is retained.

"On the fly" learning using transformers There are very few works investigating general-purpose transformer architectures that accumulate knowledge at application time. The most notable contribution in this direction is (Talmor et al., 2020), which demonstrated that pre-trained language models can be fine-tuned to incorporate new factual information and combine it with implicit knowledge learned during training. The fundamental limitation of their approach, however, is that memory retention is strictly limited by the model's context window size. In our approach, we overcome this issue by introducing a persistent world state representation.

Backpropagation Through Time alternatives Full Backpropagation Through Time (Werbos, 1990) is a standard way to train sequence models. Unfortunately, it is impractical for modern Transformer network architectures, and, even in the case of recurrent LSTM-like networks, it is notoriously unstable and memory-intensive. There have been a number of attempts to find alternatives to BPTT (Werbos, 1990). A common modification is Truncated BPTT, which cuts the gradients after a number of steps. While TBPTT sometimes leads to good results, it is also known to be biased and unstable (Mikolov et al., 2010). Tallec & Ollivier (2017) propose a modification using varied

truncation lengths and a re-weighting scheme which allows to alleviate the bias and instability, although still requiring more than one step of network unrolling, and is marked by high variance. Liao et al. (2018) propose a completely different algorithm that allows to compute the gradients with constant per-timestep memory requirements, but imposes specific assumptions about the model architecture. These restrictions make it difficult to apply the proposed algorithms to transformer-based architectures.

Our work is substantially different in that we shift the focus to the data distribution and the sampling scheme. We demonstrate that focusing on just one step is enough, if the data distribution meets certain conditions.

World state tracking A number of works (especially in the NLP domain) are concerned with the problem of world state tracking. Most notably, Weston et al. (2015) introduced the bAbI dataset to evaluate the ability of a model to quickly incorporate new data and build reasonable world state representations. EntNet (Henaff et al., 2016) is one of the most successful models for this task. However, the model is primarily LSTM-based, and, with the advent of transformer architectures, its practical value has significantly diminished. Additionally, the model has to be trained with a variable number of timesteps to achieve generalization across a variable number of timesteps, but its long-range performance still deteriorates noticeably.

In our *inductive world state representation* approach, the model is trained to perform a single step of knowledge updating. If that skill is mastered, incorporating arbitrary amounts of information becomes a matter of simply repeating the step as many times as needed. Thus, we directly train the model to generalize across timesteps instead of hoping that the network will generalize naturally.

3 PROBLEM SETTING

We are interested in modeling the evolution of a world state, where two sources of information are present: 1 – **World dynamics**, 2 – **External instructions**.

Intuitively, the first refers to changes that naturally follow from what the model knows about the world. For example, if the model is deployed as an NLP-based house assistant, it may know that every weekday, kids go to school. Therefore, when the weekend is over, a model should update its world state representation, so that the question "where are the kids" results in the answer "at school".

On the other hand, certain pieces of information cannot be realistically predicted within the scope of the model's knowledge of the world. Continuing the above example, one of the kids may get a sore tooth and go to the dentist instead of school. A reasonable house assistant system needs to be able to meaningfully incorporate such information into the world state (and to update its answer to queries like "where are the kids?" and "how much money do we owe to the insurance company?").

Other external world state updates may also reflect setting personal information (getting to know the family members, their tastes and preferences), changes in personal preferences (e.g. somebody wants to stop eating fast food), changes in one's occupation, moving to another house, and so on. While these events do not come out of nowhere and, on some level, may be predicted, their causes are out of the model's scope, and thus can be treated as arbitrary.

3.1 FORMAL SETUP

First, we need to introduce the notion of a **world state trajectory**. A **world state trajectory** W (or simply **a world**) is an abstract entity with two important properties. First, it is indexed by time. Thus, W_t represents a **world state** at time t (time can be discrete or continuous, depending on the application). We will denote the space of all possible W_t as $\mathcal{W}$.

Second, world states support the **information extraction** operation. Let $\mathcal{Q}$ be the space of all possible queries (statements about the world that may be true or false). For any query $q \in \mathcal{Q}$, the **extractor function** $f_\varepsilon(W_t, q)$ represents a binary answer to this query (whether or not the query holds in the world W at the moment of time t). When the query is provided along with its answer, it is an **instruction**, i.e. an **instruction** for a world W at time t is a pair $(f_\varepsilon(W_t, q), q)$.

Abstract entity; Representation (if different)	Interpretation
W	World trajectory
$W_t \in \mathcal{W}$; $w_t \in \mathbb{R}^n$	World state
q	A query
$v = (0/1, q)$	An instruction
$Q_t^{(i)} = \{q_k\}^i, Q_t^{(i)} \in \mathcal{Q}$	Query for world i at time t
$I_t^{(i)} = \{v_k\}^i, I_t^{(i)} \in \mathcal{I}$	Instructions for world i at time t
$f_\varepsilon : \mathcal{W} \times \mathcal{Q} \to \{0, 1\}$; $\hat{f}_\varepsilon : W \times \mathcal{Q} \to [0, 1]$	Extractor function
$f_\delta : \mathcal{W} \times \mathcal{I}^* \to \mathcal{W}$; $\hat{f}_\delta : W \times \mathcal{I}^* \to W$	Updater function

Table 1: Notation summary. It should be clear from context when we use a representation (e.g. query embedding versus an abstract query). The $\mathcal{I}^*$ notation denotes a sequence of finite instruction sets.

In the "house assistant" example, the instructions could be $I_0 = \{(True, \text{The house owner's name is John}), (True, \text{John broke up with his girlfriend})\}$, and $I_{1\text{year}} = \{(True, \text{John is still single})\}$. In this case the model should answer "no" to the query $q_{1\text{year}} = (\text{John has a wife})$. The notation is summarized in Table 1.

We can now formulate the problem. At any time t, we must provide answers to all queries based on instructions received during times $t' : t' \leq t$. Formally, given a world W, at time t' we receive a set of instructions $I_{t'} = \{(f_\varepsilon(W_{t'^{(k)}}, q^{(k)}), q^{(k)})\}, k \in \{1...K\}$. We want to compute the value $f_\varepsilon(W_t, q')$ for all $q' \in Q_t$ (all test queries at time t), using all instructions received at the time ($\{I_{t'}, t' \leq t\}$).

To make the model amenable to approximation, we assume that the worlds (world trajectories) and associated instructions come from some probability distribution $P_{W,I}$. We then define the **updater function** $f_\delta : \mathcal{W} \times \mathcal{I}^* \to \mathcal{P}(\mathcal{W})$ as $P(W_{t+1}|W_t, I_{t+1})$, where $\mathcal{P}(\mathcal{W})$ denotes the space of probability distributions over world states. In other words, the updater function outputs a distribution of world states at time $t + 1$, given a previous state and a set of incoming instructions at time t.

As further discussed in section 4, our model uses distributed representations (embeddings) w_t for W_t and approximates f_ε and f_δ with transformer modules (Vaswani et al., 2017). To initiate the process, w_0 is assumed to come from some fixed distribution. We use a point estimate to approximate the distribution $P(W_{t+1}|W_t, I_t)$. We do not explicitly model the instruction probabilities (i.e. we condition on incoming instructions, but do not try to anticipate them). However, predicting incoming instructions is a simple augmentation that may be relevant in certain applications.

3.1.1 NARROWING THE SCOPE

The problem described above is highly general. Many existing models can be interpreted in our notation. For example, traditional autoregressive language models can be interpreted as receiving a single instruction (True, "the word at position t is x") on each step, and updating the world state representation accordingly. Predicting the next word from a hidden state is equivalent to providing answers to "the word at position $t+1$ is x" queries, where x ranges over all words in the vocabulary. Time in this case runs from 1 (the first word) to n (sentence length).

In contrast, transformer language models (Vaswani et al., 2017) can be seen as receiving all instructions and queries at a single time-step. There is no recurrent world state representation to update, so in our notation, a vanilla transformer architecture has only one processing time-step. The variable-length context representation is created from the set of incoming instructions (all in the form of "the word at position k is x"), is used to answer all queries (e.g. "the masked word at position n is x"), and then discarded.

In this paper, we focus on problems that have many processing steps (as in recurrent models) as well as many instructions and queries per step. The former ensures that we can work with sequences of arbitrary length, while the latter allows us to provide dense supervision on every step, training the model to properly update its beliefs about the world.

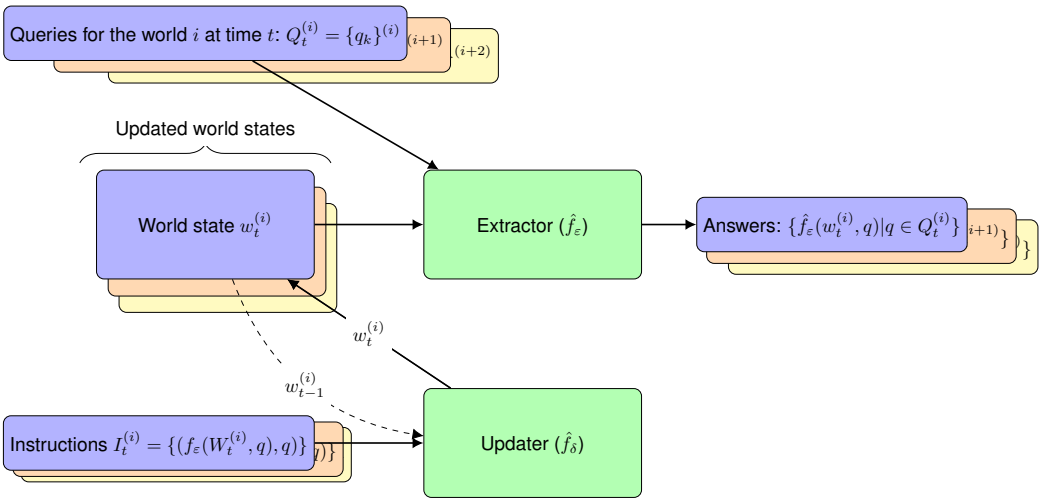


Figure 1: Updater-Extractor Architecture. The notation is introduced in the subsection 3.1. That dashed arrow indicates that no gradient is passed through the connection. The instructions and world state representation at time $t-1$ are passed to the updater which outputs a new world state representation for time t. This updated representation is then queried via the Extractor and the answers are compared to the ground truth. The gradient is **not** propagated through w_{t-1}, hence there is no need to store previous activations as in (Werbos, 1990) or similar algorithms.

4 Model Description

The model structure is illustrated in Figure 1. The model performs two types of operations on world states: *querying*, and *updating*.

For *querying*, the model receives a world state embedding w_t and a query set q. Each query encodes a specific inquiry about the state of the world. For the house assistant example above, a query may encode a question "Is John at home?". The model $\hat{f}_\varepsilon(w_t, q)$ needs to approximate the true answer $f_\varepsilon(W_t, q)$. The part of the model responsible for the query processing is called the **Extractor**. The Extractor is trained jointly with the updater, described below.

In the *updating* operation, the model needs to process a set of instructions and incorporate them meaningfully into a world state representation w_t, obtaining w_{t+1}, while also accounting for the natural world state dynamics. This part of the model is called the **Updater**, as its role is analogous to the updater function f_δ introduced in subsection 3.1.

The Updater training is mediated through the Extractor and the *querying* operation. The world states it outputs should allow the Extractor to output correct query answers. That is, if the Updater outputs $w_{t+1} = \hat{f}_\delta(w_t, I_t)$, we can obtain the Extractor predictions for a set of queries, using w_{t+1}, as $\hat{Y}_{t+1} = \{\hat{f}_\varepsilon(w_{t+1}, q)|q \in Q\}$. These predict the true answers $Y = \{f(W_{t+1}, q)|q \in Q\}$.

4.1 Architectural Decisions

Representing queries and instructions: Each query and instruction is represented as a fixed-sized vector (embedding). The way in which such embeddings are constructed is task-dependent. For example, if the task uses a knowledge-graph format, we can concatenate the embeddings of the source and target entities, together with a relation embedding (e.g. (Nickel et al., 2011)). NLP applications may use sentence or paragraph embeddings from a pre-trained language model, such as BERT (Devlin et al., 2018).

Representing world states: We split a fixed-length world state vector representation into a sequence of tokens with positional embeddings. This allows the world state to be passed into the Updater and Extractor directly, and lets the model split information into different tokens.

Representing the Extractor and the Updater: The Extractor and Updater modules are implemented as Transformers (Vaswani et al., 2017). The Updater is a transformer decoder with the world state as input and the instructions as context. I.e., the world state tokens use self-attention with themselves, and cross-attention with the instructions. In contrast, the Extractor is a transformer decoder with queries as input and the world state as context. The decoder processes many queries in parallel, but we disable self-attention between queries (since queries are independent of each other).

5 INDUCTIVE WORLD STATE REPRESENTATIONS

Knowledge internalization eliminates the need for long gradient pathways. For any recurrent model, if there is a gap between when the information is introduced and when it is first used, then keeping the gradients flowing through the gap is necessary. However, if we can ensure that the model internalizes all relevant information into its world state representation, then the need for keeping gradients between steps becomes less pressing. We call such representations **inductive**.

Since an **inductive** representation contains all relevant information for the task, it becomes intuitively reasonable to train the model on individual steps from the sequence, as opposed to full sequences. This greatly reduces the memory needed, and resolves the problem of exploding/vanishing gradients.

A natural way to ensure that the model incorporates all important information into the world state is to extensively query the world state representation on every step, making sure that no potentially useful information is lost. The architecture and the training procedure that we propose are designed to support such querying. The next section provides a theoretical background for our approach.

5.1 INDUCTIVE WORLD STATE REPRESENTATIONS: THEORY

In this section, we formalize the intuitions described in the previous section. Before introducing the result, however, we need a few additional definitions.

First, a query $q_{t'}$ is a *recall query* for an instruction v_t ($t \leq t'$) if $P(\{f_\varepsilon(q, W_t) == \text{True}\}|I_1 \dots I_t \setminus v_t \dots I'_t) = 0$ and $P(\{f_\varepsilon(q, W_t) == \text{True}\}|I_1 \dots I_t \dots I'_t) = 1$, for any instruction history s.t. $\exists t : v_t \in I_t$. In other words, the query is a recall query for an instruction if its answer is determined by whether or not the instruction was provided at an earlier step. A simple real-life example could be $q_t =$ "Did I not tell you to do your homework an hour ago?", which is a recall query for an instruction $v_{t-1} =$ ("You must do your homework now", True).

Next, we define a *thorough* query distribution. We call a query distribution *thorough* if for all instruction sequences I with nonzero probability, for all times t, for all individual instructions v_t, $\forall t' \geq t$, there is a nonzero probability of sampling a recall query $q_{t'}$ for the instruction v_t. Intuitively, it means that all incoming information may turn out to be crucial at any point of time in the future.

Lastly, we need to define the notions of **stepwise-optimal** and **sequence-optimal** models. Consider a distribution of worls and instructions $P_{W,I}$. When we pass instructions $I_1 \dots I_{t-1}$ through a model, we also obtain a distribution over world state representations $P(w_t)$ at any time t.

A model is **stepwise-optimal** if $\forall W, I \sim P_{W,I}, \forall q_t$ such that the probability of sampling q_t is positive, $P(q_t = True|w_t) = P(q_t = True|w_{t-1}, I_t)$, where $w_t = \hat{f}_\delta(w_{t-1}, I_t)$. That is, at every step, the model optimally uses all information passed through incoming instructions at time t as well as any information potentially coming from previous steps through w_{t-1}.

A **sequence-optimal** model is a model such that $\forall W, I \sim P_{W,I}, \forall t > 0, \forall q_t$ such that the probability of sampling q_t is positive, $P(q_t = True|w_t) = P(q_t = True|I_1 \dots I_t)$. In other words, it is a globally optimal model that uses all incoming information and stores all relevant information in the world state representation.

Note that these two conditions only speak about the updater part of the model, as it is the most crucial part of the model responsible for incorporating incoming information and world dynamics into the world state. An optimal extractor is such that $\hat{f}_\varepsilon(w_t, q) = P(q = True|w_t)$ everywhere.

Lemma 1 (thorough querying)**.** *Under thorough querying, any stepwise-optimal model is also sequence-optimal.*

The proof is provided in Appendix A. The main idea is that since the model may have to re-use incoming instructions at any moment, it is incentivised by stepwise(local)-optimality to keep all relevant information in the world state representation. Then, since all useful information always remains available at any local step, a locally optimal model becomes globally optimal as well.

Note that the lemma does not hold without the thorough querying assumption. For example, if we provide one bit of information on step 1 and ask to recall it on step 3, with no queries using this information on steps 1-2, a model that completely ignores the first input and outputs $w_t = 0, \forall t$ is stepwise-optimal, but not sequence-optimal.

Theory consequences Lemma 1 allows us to focus on making the model optimal on single steps (which can be done via regular gradient descent). The result guarantees that, as long as we organize a thorough training schedule, if we achieve stepwise-local optimality, the model will also be optimal globally. At present, we do not provide a formal treatment of the behaviour of near-optimal models, but it is reasonable to expect the model to gradually come closer to global optimality as it approaches stepwise local optimality, as opposed to making an abrupt jump in global performance at the moment when true local optimality is achieved. This intuition is strongly supported by our experiments.

6 TRAINING PROCEDURE

The procedure directly mirrors the theory; pseudocode is provided in algorithm 1. Having multiple queries on each step allows us to freely add recall queries, satisfying the *thorough querying* condition.

for *N outer cycles* **do**
  Data: sample K world trajectories $W^{(1)}, \ldots, W^{(K)}$, each of length T
  Initialize K world state representations $w_0^{(1)}, w_0^{(2)} ... w_0^{(K)}$;
  for t *in* $1 \ldots T$ **do**
    for k *in* $1 \ldots K$ **do**
      Sample *instructions* $I_t^{(k)}$ valid at time t.
      Obtain new world state representations
      $w_t^k = \hat{f}_\delta(STOP\_GRADIENT(w_{t-1}^k), I_t)$.
      Sample *queries* Q_t^k, obtain model predictions $\hat{Y} = \hat{f}_\varepsilon(w_t^{(k)}, Q_t^k)$.
      Compute the loss $L(\hat{Y}, f_\varepsilon(W_t^{(k)}, Q_t^k))$.
      Backpropagate the loss gradients.
    end
    if *t % update_freq = 0* **then**
      Make a gradient step, zero accumulated gradients
  end
end

Algorithm 1: Training procedure pseudocode

7 EXPERIMENTS

Through our experiments, we aim to demonstrate the practical promise of the theoretical results that we obtained, and to illustrate the breadth of potential applications of our approach. We did not aim to and did not establish SOTA results on any of the existing benchmarks. Model and hyperparameter specifications and additional details for all experiments are provided in Appendix B.

7.1 EXPERIMENT 0: LSTM RECALL

In this experiment, we directly tested the theory developed in subsection 5.1 on a single-layer LSTM architecture (Hochreiter & Schmidhuber, 1997) (in this context, the updater consists of all parts of the LSTM that update the hidden state, while the extractor is a linear layer mapping from the hidden state to a distribution over output tokens). We first describe an insightful failure case that strongly violates the thorough querying condition described in Lemma 1. Then we describe two successful

scenarios, one of which exactly satisfies the thorough query condition, while the other satisfies its weaker version, although still resulting in optimal performance.

Failure case (thorough querying violation) The task is extremely simple. We have a vocabulary of K distinct tokens (one of them is a special RECALL token). The model receives a sequence of tokens of length T. The token provided on the initial step is the *target token* for the sequence (which can not be RECALL). On all steps, if the input token is not RECALL, the model must repeat that input in its answer. If the input token is RECALL, the model must output the token memorized on the first step. The last token is always RECALL. For example, for an input sequence "4, 3, 6, RECALL, 3, RECALL", the correct output sequence is "4, 3, 6, 4, 3, 4". The task is, of course, easily learned by an LSTM model with full BPTT training, but cutting the gradients between different steps results in a complete failure; the model learns to copy the inputs, but fails the recall task. At the last step (which is always a RECALL step), the model performs at chance. This does not contradict our theory since the thorough querying condition is not satisfied. When the model receives a token other than RECALL on any step, there is no (local) incentive for the model to not forget the hidden token.

Thorough querying success cases First, in order to fully satisfy the thorough querying condition, we slightly modify the training procedure. The LSTM hidden states are not used to generate just one answer (as in standard LSTM training), but are either queried about the target token, or about the current incoming token. In this condition, the model quickly reaches ceiling performance.

The second success case is obtained by injecting *reminder noise* into the data. At any step, with a fixed probability (we tested values 0.05 and 0.01), the correct answer is replaced with the target token for the given sequence. The resulting data distribution does not strictly satisfy the conditions of Lemma 1, but it is still never beneficial to forget the target token [1]. Consequently, though the data becomes noisier, the model reaches ceiling performance (accuracy of 1 if we remove reminder noise at test time). This result suggests that the sufficient conditions in Lemma 1 can be weakened.

Simulating TBPTT Lastly, as an additional test, we vary the time at which the first recall query (for the modified LSTM training case) or the reminder noise (for the second approach) appeared in the dataset. As expected, we observe that performance drops substantially as the location of the first recall query is shifted away from the beginning of the sequence (violating thorough querying). Moreover, oftentimes the model struggles to converge and experiences rapid jumps in performance. This mirrors practical difficulties associated with using TBPTT, and illustrates that even small periods of information "irrelevance" may dramatically affect the model performance.

7.2 Experiments 1 and 2: World of Numbers and Game of Life with interventions

In the second set of experiments, we use two novel simulated tasks, *World of Numbers*, and *Game of life with interventions*, to study the model's ability to learn inductive world state representations. The *World of Numbers* task involves images of sequences of handwritten digits (from (LeCun et al., 1998)), semantically rotated forward on every step. Thus, an image depicting a sequence (0, 1, 2, 3) is changed to that depicting (1, 2, 3, 4), with all specific digit images re-sampled. The model is (sparsely) provided with and queried on values of specific pixels. In the *Game of life with interventions* task, an 8 by 8 grid world evolves according to the rules of Conway's Game of Life (Adamatzky, 2010). Queries are in the form (x, y) requesting information about the state (dead or alive) of the cell with corresponding coordinates at time t. To test whether the model can handle external changes in the state of the world, we use **interventions**: arbitrary instructions to change the state of any specific cell. The model should be able to incorporate these **interventions** into the world state and propagate them forward in time.

In both tasks, the model is trained on sequences of length 5 and tested on sequences of lengths up to 10000 with no observable drop in performance, demonstrating strong generalization ability. Surprisingly, on the sparse *World of numbers* problem, the model also learns to meaningfully integrate information over long horizons. For example, if only 5 pixel values are revealed on every timestep,

[1] The cross-entropy loss incentivises the model to keep at least some confidence allocated to the target token, even if the input does not request it.

Model	Accuracy	Model	Accuracy	Model	Accuracy
Local Attention	66.63	Longformer	69.71	BigBird	74.87
Sinkhorn Trans.	67.45	Transformer	71.40	Linear Trans.	75.30
Reformer	68.50	Sparse Trans.	71.71	Linformer	76.34
Synthesizer	69.45	**U-E (ours)**	72.52	Performer	77.05

Table 2: Pathfinder Test Accuracy. All other model results are from Tay et al. (2020).

the model still manages to recover the underlying semantic world states after approximately 100 steps. Full details on these experiments are provided in Appendix B.

7.3 Experiment 3 Pathfinder

These two experiments illustrate the breadth of potential model applications and help evaluate the model's ability to handle more challenging and practically relevant problems. We apply our model to the Pathfinder task (Houtkamp & Roelfsema, 2010; Linsley et al., 2019). Tay et al. (2020) use the Pathfinder problem as a benchmark for testing the long-range capabilities of transformer variants.

To re-interpret the static problem to fit our sequential setting, we randomly group the pixels into equally-sized chunks. At each time step, we feed a new chunk to the model, and the model is then asked to provide the values of previously observed pixels, as well as to predict the class of the instance. Compared to the more standard transformer models tested in (Tay et al., 2020), our model is more memory-efficient as it does not receive the entire image at once. Furthermore, using vanilla transformer layers, we reach a competitive accuracy (0.725), which is better than most efficient attention mechanisms (see Table 2) for long range dependencies reported in (Tay et al., 2020).

8 Discussion

Our theoretical results allow to use TBPTT in novel settings with new theoretical performance guarantees. We believe that our result has broad implications, since the idea of focusing on one-step predictions was often voiced before, but was usually rejected due to reasons that we resolve in the present contribution. For example, in Reinforcement Learning, next-step prediction was previously discussed but rejected (Gregor et al., 2019) because of the concern that long-term dependencies may be irrelevant for short-term prediction (this concern is resolved through thorough querying). Similarly, in NLP, most works try to avoid situations requiring BPTT or TBPTT because of practical issues (see, e.g. (Rae et al., 2019), section 3.2), lack of stability, and theoretical guarantees, all of which are resolved in our paper. **Overall, restructuring training to satisfy the thorough querying assumption makes TBPTT practical, theoretically justified, and empirically reliable**.

We test our theoretical results both on a simple LSTM and on a modified transformer architecture. Empirical results strongly support the theory, demonstrating that 1) strong violations of sufficient conditions that we identified lead to TBPTT failures 2) when sufficient conditions are only partially satisfied (see Experiment 0), high performance can still be achieved, suggesting that with further theoretical development, our sufficient conditions may be weakened 3) competitive results on challenging long-horizon tasks can be obtained by applying our method to transformer architectures, making our approach applicable in a wide range of circumstances.

There are two limitations of our approach. First, the data for our model must be structured differently than in traditional sequence-to-sequence training. In some cases (as in the Pathfinder problem), simple re-interpretation is sufficient and leads to good performance, but in other cases (e.g. bAbI (Weston et al., 2015)) the dataset needs to be significantly changed. **Second**, for the tasks that require rote memorization (such as Game of Life, where knowing one fact (pixel activation) tells very little about whether or not other facts are true), a very high knowledge retention rate is required. Otherwise, the model knowledge will quickly deteriorate. We, therefore, believe that our approach is best suited for domains allowing for rich common sense reasoning, where different pieces of information are highly entangled, allowing the model to reconstruct forgotten knowledge from what it still remembers.

Ethics Statement Our work targets a problem related to sequence processing and NLP, and ultimately could advance the systems that can learn via natural interactions at application time. It may dramatically advance the repertoire of tasks that personal assistants and dialogue agents may accomplish, which could improve quality of life, but may also lead to some societal problems, such as unemployment. Overall, however, we hope that potential benefits outweigh the risks.

REFERENCES

Andrew Adamatzky. *Game of life cellular automata*, volume 1. Springer, 2010.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines, 2014.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 2016.

Karol Gregor, Danilo Jimenez Rezende, Frederic Besse, Yan Wu, Hamza Merzic, and Aaron van den Oord. Shaping belief states with generative environment models for rl. *arXiv preprint arXiv:1906.09237*, 2019.

Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun. Tracking the world state with recurrent entity networks. *arXiv preprint arXiv:1612.03969*, 2016.

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

Roos Houtkamp and Pieter Roelfsema. Parallel and serial grouping of image elements in visual perception. *Journal of experimental psychology. Human perception and performance*, 36:1443–59, 12 2010. doi: 10.1037/a0020248.

Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *arXiv preprint arXiv:2101.01169*, 2021.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models, 2020.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Renjie Liao, Yuwen Xiong, Ethan Fetaya, Lisa Zhang, KiJung Yoon, Xaq Pitkow, Raquel Urtasun, and Richard Zemel. Reviving and improving recurrent back-propagation. In *International Conference on Machine Learning*, pp. 3082–3091. PMLR, 2018.

Drew Linsley, Junkyung Kim, Vijay Veerabadran, and Thomas Serre. Learning long-range spatial dependencies with horizontal gated-recurrent units, 2019.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.

Nikita Nangia and Samuel R. Bowman. Listops: A diagnostic dataset for latent tree learning, 2018.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *Icml*, 2011.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling, 2019.

Corentin Tallec and Yann Ollivier. Unbiasing truncated backpropagation through time. *arXiv preprint arXiv:1705.08209*, 2017.

Alon Talmor, Oyvind Tafjord, Peter Clark, Yoav Goldberg, and Jonathan Berant. Teaching pre-trained models to systematically reason over implicit knowledge. *arXiv preprint arXiv:2006.06609*, 2020.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*, 2020.

Aaron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional image generation with pixelcnn decoders, 2016.

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

Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.

APPENDIX A. PROOF OF LEMMA 1 (THOROUGH QUERYING)

ASSUMPTIONS, NOTATION RECAP, AND THE LEMMA STATEMENT

We assume that there is a probability distribution over world trajectories, instructions, and queries $W, I, Q \sim P_{W,I,Q}$. Sampled instructions I contain instruction sets for every step, i.e. $\{I_1, I_2, I_3 \dots I_n \dots\}$. We assume that the space of all possible instruction sequences is countable (and therefore we must assume that we are working with sequences of finite length or, for convenience, infinite sequences with redundant (repeating) tails).

Notation details for the proof. For convenience, we denote $\{I_1 ... I_n\}$ as $I_{1...n}$. Additionally, we use $I_{1...n} \setminus v_t$ to denote $\{I_1, \dots, I_t \setminus v_t, \dots, I_n\}$. Q contains queries sampled for every step. For simplicity, we assume that we only sample individual queries (as opposed to query sets) for every step. That is, $Q = \{q_1, q_2, q_3, \dots\}$. In the proof, we use $P(q = True|I_{1...n})$ or $P(q|I_{1...n})$ instead of $P(\{f_\varepsilon(q, W_t) = \text{True}\}|I_{1...n})$.

A query $q_{t'}$ is a *recall query* for an instruction v_t ($t \leq t'$) if $P(\{f_\varepsilon(W_t, q_{t'}) = \text{True}\}|I_1 \dots I_t \setminus v_t \dots I'_t) = 0$ and $P(\{f_\varepsilon(W_t, q_{t'}) = \text{True}\}|I_1 \dots I_t \dots I_{t'}) = 1$, for any instruction history s.t. $\exists t : v_t \in I_t$.

A query distribution is called *thorough* if for all instruction sequences I with nonzero probability, for all times t, for all individual instructions v_t, $\forall t' \geq t$, there is a nonzero probability of sampling a recall query $q_{t'}$ for the instruction v_t. Intuitively, it means that all incoming information may turn out to be crucial at any point of time in the future.

Next, fix a model (a pair $\hat{f}_\delta, \hat{f}_\varepsilon$). We define random variables $w_t = \hat{f}_\delta(w_{t-1}, I_t)$ for all $t > 0$. We assume that w_0 is either a constant or a random variable sampled independently of I, W and Q. We also assume the $\hat{f}_\delta, \hat{f}_\varepsilon$ output point estimates and are deterministic.

A model is **stepwise-optimal** if $\forall W, I \sim P_{W,I}, \forall t > 0, \forall q_t$ (such that the probability of sampling q_t is positive), $P(q_t = True|w_t) = P(q_t = True|w_{t-1}, I_t)$.

A **sequence-optimal** model is a model such that $\forall W, I \sim P_{W,I}, \forall t > 0, \forall q_t$ (such that the probability of sampling q_t is positive), $P(q_t = True|w_t) = P(q_t = True|I_1 \dots I_t)$.

Lastly, the statement of Lemma 1 is as follows: *under thorough querying, any stepwise-optimal model is also sequence-optimal*.

THE PROOF

Assume that the model $\hat{f}_\delta$ is stepwise-optimal. Then the model is also sequence-optimal, by induction.

Base Note that w_0 is a world state representation before any information is provided. It is either a fixed constant for the initial world state representation or a random variable drawn from some fixed initialization distribution. That is, w_0 is independent from W, I, and Q. Therefore, at $t = 1, \forall q_1, p(q_1 = True|I_1, w_0) = p(q_1 = True|I_1)$. Then, by stepwise-optimality, we have $P(q_1 = True|w_1) = P(q_1 = True|I_1)$, which is the condition for sequence optimality at $t = 1$.

Step Assume that sequence optimality holds for all $t \leq n \in \mathbb{N}$. We want to show that at $t = n+1$, the condition holds as well, i.e. that $\forall W, I \sim P_{W,I}$, $\forall q_{n+1}$ with positive sampling probability, $P(q_{n+1}|w_n, I_{n+1}) = P(q_{n+1} = True|I_{1,\dots n+1})$.

Let's consider an arbitrary q_{n+1} with a nonzero sampling probability.

First, note that by stepwise-optimality, we have $P(q_{n+1} = True|w_{n+1}) = P(q_{n+1} = True|w_n, I_{n+1})$. Therefore, it remains to show that $P(q_{n+1} = True|w_n, I_{n+1}) = P(q_{n+1} = True|I_{1\dots n+1})$.

Fix any instruction $v_t \in I_{1\dots n}$. Since sampling is thorough, there exists a recall query q'_n for v_t with a nonzero sampling probability. I.e. a query at time n s.t. $P(q'_n = True|I_{1\dots n} \setminus v_t) = 0$ and $P(q'_n = True|I_{1\dots n}) = 1$. Fix any such query q'_n.

By the inductive assumption, $P(q'_n = True|w_n) = P(q'_n = True|I_{1\dots n})$, which is equal to 1 by definition of the recall query. But then notice that, again, by definition, the event $\{q'_n = True\}$ is equivalent to the event that the instruction v_t is in the history $I_{1\dots n}$. Therefore, $P(v_t|w_n) = 1$.

Next, we want to show that adding I_{n+1} to the conditioning set of $P(v_t|w_n)$ will not change the probability. First, notice that $P(w_n, I_{n+1}) > 0$, since both w_n and I_{n+1} are coming from the sequence of instructions I that was sampled (and hence had positive probability)[2]. Consequently (since $P(w_n, I_{n+1}) = P(I_{n+1}|w_n)P(w_n)$), $P(w_n, I_{n+1}) > 0$ as well. Therefore, we can condition on w_n, I_{n+1} without creating a contradiction, as well as divide by $P(I_{n+1}|w_n)$. Therefore, note that $1 = P(v_t|w_n, I_{n+1}) + P(\bar{v_t}|w_n, I_{n+1})$. But $P(\bar{v_t}|w_n, I_{n+1}) = \frac{P(\bar{v_t} \cap I_{n+1}|w_n)}{P(I_{n+1}|w_n)} \leq \frac{P(\bar{v_t}|w_n)}{P(I_{n+1}|w_n)} = \frac{1-P(v_t|w_n)}{P(I_{n+1}|w_n)} = 0$. Hence, overall, $P(v_n|w_n, I_{n+1}) = 1$.

Then, if we come back to the original query q_{n+1}, notice that $P(q_{n+1}|w_n, I_{n+1}) = P(q_{n+1} \cap v_n|w_n, I_{n+1}) + P(q_{n+1} \cap \bar{v_t}|w_n, I_{n+1})$, but $P(q_{n+1} \cap \bar{v_t}|w_n, I_{n+1}) \leq P(\bar{v_t}|w_n, I_{n+1}) = 0$. Thus, overall, $P(q_{n+1}|w_n, I_{n+1}) = P(q_{n+1} \cap v_t|w_n, I_{n+1})$.

[2]Note that we use our countability assumption here: because of it, we can have a discrete probability defined over sequences of instructions and so that any sampled instruction has positive probability.

We can, therefore, proceed as follows:

$$
\begin{aligned}
&P(q_{n+1}|w_n, I_{n+1}) = \\
&P(q_{n+1} \cap v_t|w_n, I_{n+1}) = \\
&P(q_{n+1}|v_t, w_n, I_{n+1})P(v_t|w_n, I_{n+1}) = \\
&P(q_{n+1}|v_t, w_n, I_{n+1})
\end{aligned}
$$

We have shown that we can add any individual instruction $v_t, t \leq n$ from the conditioning set. Notice that we can repeat the reasoning with any other instruction from the history $I_{1\dots n}$. Moreover, the reasoning holds exactly analogously if we replace I_{n+1} with $I_{n+1}, v_{t_1}, v_{t_2}, \dots$, as long as all v_{t_i} are instructions the true history $I_{1\dots n}$.

Therefore, we can add all individual instructions $v \in I_{1\dots n}$ into the conditioning set. In other words, we get the following result: $P(q_{n+1}|w_n, I_{n+1}) = P(q_{n+1}|w_n, \{v_t : v_t \in I_{1\dots n}\}, I_{n+1}) = P(q_{n+1}|w_n, I_{1\dots n+1})$!

Since w_n is a deterministic function from $I_{1\dots n}$ (i.e. $w_n = \hat{f}_\delta(\hat{f}_\delta(\dots \hat{f}_\delta(\hat{f}_\delta(w_0, I_1), I_2), \dots), I_{n-1})$, we have $P(q_{n+1}|w_n, I_{1\dots n}) = P(q_{n+1}|I_{1\dots n+1})$, which completes the proof. ■

DISCUSSION

The conditions for Lemma 1 are only sufficient conditions and can be relaxed. For example, at a certain time step, if there are past instructions that we know will not be relevant to any future queries, or which have been superseded by later instructions, those instructions do not need to have recall queries.

During training, instead of asking all relevant recall queries at every step, we can of course sample them probabilistically, as this does not change the objective function. This allows for much faster training.

APPENDIX B. EXPERIMENT DETAIL

In this section we provide a more detailed description for all experiments introduced in the paper. The code and replication instructions for all experiments is provided along with the submission. If the submission is accepted, we will make the code publicly available.

HARDWARE

All reported experiments were run on an NVIDIA 1080Ti GPU.

HYPERPARAMETER TUNING METHODOLOGY

We mostly chose hyperparameters close to conventional "best practices". We did not perform any automated hyperparameter optimization (due to computational costs), but we did adjust learning rates and network sizes manually to find adequate learning regimes.

CURRICULUM LEARNING

We use a basic form of curriculum learning to speed up training for the Pathfinder task: the number of steps in each world trajectory starts out truncated, and the model must achieve recall accuracy beyond a certain threshold for some number of training iterations in order for the world trajectory to be lengthened. The rationale for doing this is that at initialization, the model is far from able to retain information at any step. A model which fails to retain information at a certain step will almost certainly fail to recall that information at later steps as well. Therefore, the gradient from late steps will not help in retaining things that were already forgotten by the time these late steps are reached. In other words, under curriculum learning, the model is trained to retain information for progressively larger numbers of time steps.

EXPERIMENT 0 (LSTM RECALL), FAILURE CASE DETAIL

Data We have a vocabulary of K distinct tokens (one of them is a special RECALL token). The model receives a sequence of tokens of length T. The token provided on the initial step is the *target token* for the sequence (it can not be a "RECALL" token). On all steps, if the input token is not RECALL, the model must repeat that input in its answer. If the input token is RECALL, the model must output the token memorized on the first step. The last token is always RECALL. For example, for an input sequence "4, 3, 6, RECALL, 3, RECALL", the correct output sequence is "4, 3, 6, 4, 3, 4". We used 10 different tokens and a sequence length of 10 (we also tried a sequence length of 25, obtaining the same results with no qualitative difference in conclusions). We randomly generated a sequence of numbers and then flipped input tokens in every position (except the first) to RECALL with a probability of 0.3.

Architecture detail We used a single-layer LSTM architecture instead of the full transformer-based Updater-Extractor architecture. In this case, the Extractor is represented by a single matrix, mapping the hidden state to the distribution over answers. The Updater comprises all other parts of the network, responsible for updating the hidden state and the cell states. In this case, the Extractor does not receive the query separately from the world state. In other words, the question that the model needs to answer has to be encoded in the world state representation along with the actual information about the world. This architectural decision is common to most sequence-to-sequence models. Unfortunately, for such models, querying the world state necessarily changes it. As explained in the next section, this prevents us from doing thorough querying. The absence of thorough querying, in turn, results in the model's failure to learn the task under TBPTT.

Training detail We train the model using AdamW (Loshchilov & Hutter, 2017) with a batch size of 128, learning rate of 1e-4 and default parameters otherwise. We use a standard Cross-Entropy loss function. The LSTM has the hidden state dimension of 64.

The full BPTT model converges to the optimum under two minutes, while the TBPTT model stays at chance and makes no progress in an hour.

Failure additional discussion This failure does not contradict our theoretical results, as the thorough querying condition is not satisfied.

Indeed, when the model receives a token other than RECALL on any step after the first one, there is no (local) incentive for the model to not forget the information about the hidden token. In traditional sequence-to-sequence modeling, the set of queries is exactly the same on every step: $\{y_t = A?$, $y_t = B?$, $y_t = C?$... $\}$, i.e. "is the output token at time t equal to A/B/C/etc.". Because of that, for an instruction sequence $4, 6$, after the instruction 6 is provided at the second timestep, there is no dependence between the value of the number at the first timestep and the current output. Therefore, the model is not incentivised to remember this information.

This illustrates the crucial difference between traditional sequence-to-sequence training and our approach: in sequence-to-sequence training, the set of queries is rigid; the same queries are answered on every step, and if a certain piece of information is not immediately relevant given the incoming instructions (the input is not requesting it), there is no local incentive to keep it in the world state representation. Unless the representation is of infinite capacity, non-thorough querying actively incentivises forgetting past information.

This negative result highlights another limitation of traditional sequence to sequence training: the boundary between the instructions and queries is blurred. The inputs are used to both provide information about the world and to request the necessary information out of the network (like in our recall task). The absence of independent mechanisms for probing the world state makes it difficult to investigate the model's beliefs about the world or to explicitly train it to change those beliefs. That is, the information about the consequences of any incoming information must be spoon-fed, one element at the time, as there is usually only one answer at any timestep.

EXPERIMENT 0 (LSTM RECALL), SUCCESS CASES DETAIL

Modified LSTM success case In order to fully satisfy the *thorough querying* requirement, we need to disentangle providing instructions and querying, which requires a slight modification to the architecture and the data itself (there is no reason to have RECALL inputs anymore, since querying is done independently).

To satisfy the thorough query condition, the hidden states that are generated should not be used to generate just one answer (as in traditional sequence-to-sequence training), but should rather be queried either about the target token, or about the current incoming token.

We tried to achieve that with minimal changes to model architecture and capacity. To allow for independent querying, instead of a linear mapping from the hidden state directly to output tokens (as in traditional LSTM), we first concatenate the hidden state with the query encoding and then pass the result through an MLP with one hidden layer (width 64) to obtain the answer. There are only two different queries: "recall" and "repeat". Since in this case the updater does not know what the hidden state is going to be queried about, it always remains optimal to remember the target token (apart from encoding the current input).

The setting described above fully satisfies the *thorough querying* assumption. Consequently, the TBPTT model quickly reaches ceiling performance (under 2 minutes). Full BPTT converges 2 times faster, but requires n times more memory, where n is the sequence length.

Reminder noise success case For the *reminder noise injection* success case, all model and training parameters remained the same. The full BPTT model converges to optimality under two minutes, while the TBPTT model takes longer. With the 0.1 noise injection probability, the model takes slightly less than 10 minutes to converge to optimality, while with the reminder noise injection probablity of 0.05 the models take from 45 minutes to an hour to converge to optimality.

Recall gaps Lastly, introducing "gaps" in recall queries (timesteps with recall query probability equal to zero) or similar gaps for reminder noise severely hinders model performance. This is natural, since such gaps violate the thorough querying assumption.

Overall, all the results above indicate that there is a relationship between how strongly the thorough querying assumptions are violated and how hard it it for the TBPTT model to find the optimal solution.

EXPERIMENT 1 (WORLD OF NUMBERS) DETAIL

The first experiment using our transformer-based architecture is the "world of numbers" problem. In this setting, the world states are n-tuples of handwritten digit images. The queries are in the form (e_1, e_2, r), where e_1 and e_2 represent pixel coordinates and r represents the index of the image. For example, the query $(14, 7, 3)$, requests the pixel at $x = 14$, $y = 7$ from the third image in the n-tuple. We use a binarized version of the MNIST dataset (LeCun et al., 1998) to obtain the images. This problem representation mimics the structure of working with a knowledge graph: r corresponds to relation numbers, and e_1 and e_2 denote different entities.

The world dynamics are simple: at each timestep, the n-tuple is "semantically" rotated forward. That is, if the initial tuple at t_0 comprises images of digits 1, 2, 3, 4, and 5, the tuple on the step t_1 will comprise the (newly sampled) images of digits 2, 3, 4, 5, and 6 (see Figure 2).

On every step, the model is provided with information about some of the pixel values and is queried about the values of other pixels. To make the task more challenging, we only provide very limited information: we sample (uniformly at random) between 0 and 75 pixels out of 2352 (i.e. $28^2 \cdot 3$) as instructions on every step. That is, the data that the model receives is very sparse, so it has to rely on its knowledge of the world dynamics.

The goal of this simple experiment is to investigate the model's capacity to follow instructions, learn global world dynamics, and, lastly, its ability to handle uncertainty.

Model architecture and training details In this experiment, the updater consists of 4 transformer layers, with 2 heads, 1024 for the hidden dimension of fully connected blocks, and 32 for the

(a) (b)

Figure 2: "World of numbers" problem setting. a) World state transition example. b) Providing sparse information. Left column - true world. Middle - information given at step 1 (black pixels are not shown to the model). Right - the model's predictions about the world after only receiving the information in the middle column. Since very little information was given, the model struggles and predicts generic shapes.

query/key/value dimensions. The extractor is the same but only uses 2 transformer layers, with an additional full-connected layer on top for the output. The world state consists of 8 tokens. The batch size is 128 and the model is trained for 50000 iterations, using Adam with a learning rate of 1e-3, which converges in around 5 hours on our hardware.

LEARNING WORLD DYNAMICS & EVALUATING TRAJECTORY STABILITY

First, we test the model's ability to learn the world dynamics. We train the model by providing values of 0-500 pixels at $t = 1$, and then providing additional values of 0-75 pixels on every step thereafter. Additionally, on every step, the model is queried about the value of various pixels, both observed and unobserved (75 pixels total).

We advance the worlds eight time steps forward during training, without propagating gradients between different steps.

The first important result is that the model quickly reaches ceiling performance when it comes to incorporating pixel value information given directly (i.e. if a pixel value was provided at time t, the model is able to remember that value). That means that the updater and extractor learn to meaningfully coordinate their actions, using world state representations to store/communicate all relevant information.

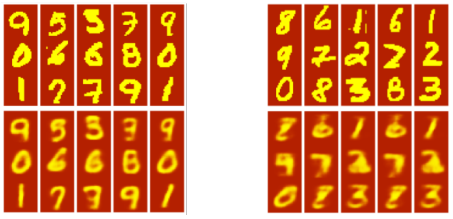
Learning semantic world dynamics More importantly, the model learns to predict appropriate values for unobserved pixels, extrapolating from the available information. The model also learns the semantics of the world dynamics. That is, even if no information is provided during some steps (at application time), the model appropriately keeps track of the state of the world. For example, if the first step provided ample information about images of digits (5, 6, 7), the model infers that the third step must contain pictures of digits (7, 8, 9) and predicts appropriate generic shapes of numbers (7, 8, 9), even if we provide no information at all after the first step (and hence the model can not be precise in guessing the specific handwritten instances of the numbers it must predict). See Figure 3 for an illustration.

EXPERIMENT 2: GAME OF LIFE WITH INTERVENTIONS

The "world of numbers" experiment handles worlds with a relatively small number of possible "semantic" world states (even though the actual images may vary greatly, in total, there are only 10 different image classes). In this experiment, we test the model's ability to handle much more complex scenarios, where the space of possible world states is exponentially large and requires modeling local entity-to-entity interactions between timesteps.

For this purpose, we created a **Game of Life with interventions** environment. In this experiment, an 8 by 8 grid world evolves according to the rules of Conway's Game of Life (Adamatzky, 2010). Queries are in the form (x, y) requesting information about the state (dead or alive) of the cell with corresponding coordinates at time t.

To test whether the model can handle arbitrary external changes in the state of the world, we introduce the notion of an intervention: an arbitrary (not predictable from world dynamics) instruction



(a) (b)

Figure 3: Trajectory stability. a) Top-true world state on step 1. Bottom-model beliefs on step 1, after 1500 out of 2352 pixels values are provided. Notice that the model captures each handwritten digit style: for example, ones are tilted at a different angle, reflecting the data. b) A world at t=50, where again, all information is given on the first step, with no input afterwards. The model has no information about specific digit instances, but knows (from step 1) about their identities. Therefore, the model predicts generic digit shapes with correct identity. Notice that all digits having the same identity are reconstructed identically. Notably, rolling a world, 10, 1000 or a 10000 steps forward with no input information results in visually identical reconstruction, showing that the model retains its world state knowledge across apparently arbitrary horizons.

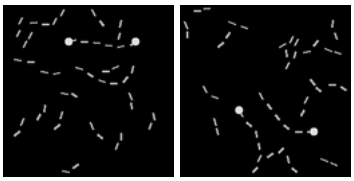


Figure 4: Pathfinder challenge problem example. The task is to determine whether the big dots lie on the same path (left) or on different paths (right). This task proved to be fairly challenging for a number of models (Houtkamp & Roelfsema, 2010; Tay et al., 2020).

to change the state of any specific cell. The model should be able to instantly incorporate these **interventions** into the world state and meaningfully propagate them forward in time.

Model architecture and training In this experiment, the updater consists of 4 transformer layers, with 4 heads, 2048 for the hidden dimension of fully connected blocks, and 256 for the query/key/value dimensions. The extractor is the same but only uses 2 transformer layers, with an additional fully-connected layer on top for the output. The world state consists of 32 tokens. The batch size is 256 and the model is trained for 60000 iterations, using Adam with a learning rate of 1e-4, which takes about 3 days on our hardware.

Experiment 2 (Game of life with interventions): additional discussion

We successfully train two models, one on worlds without interventions and on worlds with variable numbers of interventions on each time step. In both cases, we provide all pixels at the first time step. In the worlds without interventions, we do not provide any inputs in subsequent steps, and in the world with interventions, we provide inputs corresponding to these interventions.

As before, the models are able to continue correctly updating the model over an arbitrary number of steps during test time (we tested the model by rolling the world up to 10000 steps, with no drop in performance).

To put these results in perspective, it may be useful to compare them to the toy dataset results in Henaff et al. (2016). In that paper, the authors track a world state on a 10x10 grid with 2 agents, each of which have 4 possible states (facing top/down/left/right) and can be located in any square. The agents also do not affect each other in any way. This world has $16 \cdot 10^4 < 2^8$ possible states.

In contrast, the game of life on an 8x8 grid has 2^{64} possible initial states and involves learning non-trivial agent interaction dynamics. In addition, interventions pose an additional challenge as they require the model to break the natural world dynamics upon request.

The problem dynamics considered in Henaff et al. (2016) can be exhaustively learned even by a very moderately sized network. In the Game of Life with interventions, exhaustively memorizing 2^{64} state trajectories is not feasible, so the model has to internalize the rules in order to perform well.

EXPERIMENT 3: PROGRESSIVE PATHFINDER

To test the model in more challenging setting, we applied our model to the Pathfinder problem (Houtkamp & Roelfsema, 2010). This task was re-introduced as a machine learning benchmark by Linsley et al. (2019), who found that CNNs were inefficient at learning the requisite long-range dependencies to solve the problem. (Tay et al., 2020) use the Pathfinder problem as one of the benchmark tasks for testing transformers and their ability to handle long range information(see Figure 4).

To re-interpret the static problem in order to fit our sequential setting, we randomly group the pixels of an instance into a number of equally-sized partitions. At each time step, we feed a new partition to the model, and the model is then asked to provide the values of previously observed pixels, as well as to predict the class of the instance. In order to solve the problem, the model must find a world state representation that is flexible enough to handle the reception of information in an unordered manner, yet structured enough to readily decode the relevant long-range path information in order to correctly classify the instance.

In this setting, the model functions similarly to PixelCNN (van den Oord et al., 2016) (though note that PixelCNN is not generally used for classification). However, traditional PixelCNN models require fixed-order inputs and are difficult to train due to their sequential nature. We are able to take advantage of the parallelism inherent in transformer-based architectures to avoid these drawbacks.

In comparison to the more standard transformer models tested in Tay et al. (2020), our model is more memory-efficient as it does not need to receive the entire image at once. Furthermore, using vanilla transformer layers, we reach a competitive accuracy (0.725), which is better than most efficient attention mechanisms for long range dependencies reported in Tay et al. (2020). This demonstrates that mixing parallelism and recurrence can be more effective than simply trying to use an extremely large context window.

Model architecture and training In this experiment, both the updater and extractor consist of 4 transformer layers, with 4 heads, 2048 for the hidden dimension of fully connected blocks, and 128 for the query/key/value dimensions; the extractor has an additional fully-connected layer on top for the output. The world state consists of 16 tokens. The model is trained for 130 epochs, using Adam with a learning rate of 3e-4 and a batch size of 64, which takes about 2 days on our hardware.

EXPERIMENT 3.B: LISTOPS

We also applied our model to the Listops problem (Nangia & Bowman, 2018), which is another benchmark used in Tay et al. (2020) which involves parsing a sequence of nested operations on lists of digits and computing the end result. In our setting, we break the sequence into chunks of 250 tokens and feed in the chunks sequentially.

Model architecture and training In this experiment, the updater consists of 4 transformer layers, with 8 heads, 2048 for the hidden-dimension of fully-connected blocks, and 64 for the query/key/value dimensions. The world state consists of 512 tokens. The model is trained using Adam with a learning rate of 3e-4 and a batch size of 64. The training takes around 8 hours on our hardware.

APPENDIX C: EXTERNAL RESOURCES

- All models were implemented in PyTorch - BSD
  https://pytorch.org/
- MNIST data - CC BY-SA 3.0
  http://yann.lecun.com/exdb/mnist/

- Pathfinder data - MIT License
  https://github.com/google-research/long-range-arena [3]
  https://github.com/drewlinsley/pathfinder

[3] We use the "hard" dataset, which is the same as in their reported experiments.