Explain My Surprise: Learning Efficient Long-Term Memory by Predicting Uncertain Outcomes

Artyom Sorokin
AIRI, MIPT
Moscow, Russia
asorokin@airi.net

Nazar Buzun
AIRI
Moscow, Russia
buzun@airi.net

Leonid Pugachev
MIPT
Dolgoprudny, Russia
puleon@mail.ru

Mikhail Burtsev
AIRI, MIPT
Moscow, Russia
burtsev@airi.net

Abstract

In many sequential tasks, a model needs to remember relevant events from the distant past to make correct predictions. Unfortunately, a straightforward application of gradient based training requires intermediate computations to be stored for every element of a sequence. This requires to store prohibitively large intermediate data if a sequence consists of thousands or even millions elements, and as a result, makes learning of very long-term dependencies infeasible. However, the majority of sequence elements can usually be predicted by taking into account only temporally local information. On the other hand, predictions affected by long-term dependencies are sparse and characterized by high uncertainty given only local information. We propose MemUP, a new training method that allows to learn long-term dependencies without backpropagating gradients through the whole sequence at a time. This method can potentially be applied to any recurrent architecture. LSTM network trained with MemUP performs better or comparable to baselines while requiring to store less intermediate data.

1 Introduction

Two dominating approaches for memory augmentation in deep learning include recurrent neural networks [1; 2; 3; 4; 5; 6; 7; 8; 9]) and Transformers [10; 11; 12; 13; 14; 15]. Historically, recurrent networks and Transformers have been primarily developed for applications in natural language processing, which is characterized by strong dependencies between closely located elements. On the other hand, in the reinforcement learning setting, the agent often has to remember only a few bits of information but for a very long time [16].

Evolution of neural architectures tackling the problem of learning long-term dependencies follows the path of increasing size and complexity [6; 8; 17]. For example, Transformers[18] and memory augmented neural networks [7] perform much better than the classical LSTM in tasks with long-term temporal dependencies. Still, even gigantic language models like GPT-3 [19] process sequences only up to 2048 elements due to quadratic complexity of self-attention. So, these architectures are much more demanding in terms of the memory capacity and computational power required for training.

To learn a long-term dependency between a distant observation in the past and agent's actions in the current state, both the recurrent networks trained with back propagation through time and Transformers need to store gradients for all intermediate steps. Taking into account, that the minimum distance between useful information in the past and the moment of its utilization can be measured in thousands or millions of time steps, such solutions seem unrealistic given the current hardware capabilities. We propose a training method that allows an agent to find and store dependencies between temporarily distant events without the need to process intermediate steps.

Our contributions are the following:

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

- We propose a new formulation for the problem of memory learning as maximisation of conditional mutual information between memory state and future outcomes.
- We introduce an original method MemUP (**Mem**ory for **U**ncertainty **P**rediction) for efficient long-term memory training by minimizing uncertainty of predictions for states where local information is not enough.
- We show that implementation of our method for recurrent neural networks is able to learn long-term dependencies better than baselines while propagating gradients only through a small fractions of a sequence in one optimization step. Notably, the applications of MemUP are successful in supervised learning tasks with different loss functions as well as in reinforcement learning tasks.

2 Memory to explain surprise

The main idea of our approach is very simple. Find the most surprising events or elements of a sequence and train a memory to detect and store information that allows to explain out these events. Below we formulate this idea in the framework of information theory.

Lets consider a sequence τ with inputs $\{x_t\}_{t=0}^{T}$ and targets $\{y_t\}_{t=0}^{T}$. A model is trained to predict targets given inputs. An ideal memory could just store all previous inputs $\{x_i\}_{i=0}^{t}$ at each step t and predict y_t from them, but it is too computationally expensive. In practice a memory model g_θ has to selectively store some aggregation of this sequence in a memory state $m_t = g_\theta(x_t, m_{t-1})$. Then it should be decided what to write and what to remove from the memory.

Estimating past information importance. Consider an arbitrary time step k. The past element x_t has useful information about the target y_k, if y_k depends on x_t given current x_k, i.e. $p(y_k|x_k, x_t) \neq p(y_k|x_k)$. In this case, we can say that there is a temporal dependency with the length of $k - t$ steps, that starts at step t and ends at step k. The strength of temporal dependency or importance of remembering x_t can be measured as an amount of change in the distribution of y_k given a knowledge of x_t . Multiplication of the both parts of the inequality above by $p(x_t|x_k)$ and application of KL-divergence as a measure of the difference between distributions leads to the conditional mutual information as a measure of x_t importance:

$$I(y_k; x_t|x_k) = \mathbb{E}_{x_k}\mathcal{KL}[p(y_k, x_t|x_k)\|p(y_k|x_k)p(x_t|x_k)], \tag{1}$$

where x_k is taken from sequences stored in some dataset or generated by an agent over interaction with the environment.

Objective function for memory training. Then, the problem of learning what to store in the memory can be defined as maximization of the mutual information between memory states and prediction targets with respect to the parameters θ:

$$\max_{\theta} \sum_{k=t}^{T} I(y_k; m_t = g_\theta(x_t, m_{t-1})|x_k). \tag{2}$$

The sum in eq. 2 allows us to process each memory update in a separate gradient step. As accounting for all future steps prevents loss of information that can be helpful in the distant future. Unfortunately, this still requires the entire remaining (future) sequence to be processed for every update and has the complexity $O(T)$ with respect to the sequence length.

Reducing the cost of memory training. Fortunately, updating the memory for all future steps is not necessary for the majority of real-world tasks. As the distance between events increases, the number of dependencies also decreases. This phenomenon is sometimes called locality of reference [20]. In other words, information from the greater part of past inputs may be useless for y_k prediction given x_k. Accordingly, most of the elements in the sum (eq. 2) will be close to zero. Therefore, depending on the structure of the problem, we could vary the number of elements in the sum, leaving only those for which long-term memory is critical for correct predictions. The main problem is to find such steps in the sequence.

Lets assume that the ideal memory m_t^* that stores all possible information from the past is available. Then the conditional mutual information at a step t:

$$I(y_t, m_t^*|x_t) = H(y_t|x_t) - H(y_t|m_t^*, x_t), \tag{3}$$

shows the potential utility of memory for prediction at this step. When the number of elements in the sum in eq. 2 is minimized to reduce computations, the steps with the highest value of mutual information from eq. 3 should be preserved to assure good quality of prediction. Naturally, we cannot estimate $I(y_t, m_t^*|x_t)$ directly, because m_t^* is not available to us.

As seen from the eq.3, m_t^* participates only in the second entropy, which indicates how well one can predict the future in the presence of ideal memory and ideal model. On the other hand, the local entropy $H(y_t|x_t)$ can be easily estimated. A value of local entropy would be sufficient for our task if we could use it to determine the ordering for values of the potential memory utility $I(y_t, m_t^*|x_t)$. Specifically, for every pair of elements i, j if $H(y_i|x_i) > H(y_j|x_j)$ then $I(y_i, m_i^*|x_i) > I(y_j, m_j^*|x_j)$. This becomes true under one of two possible conditions: (1) $H(y_t|x_t) \gg H(y_t|m_t^*, x_t)$ or (2) $H(y_t|m_t^*, x_t) \sim \epsilon$. The first condition means that given a perfect memory and a model, one could make high-quality predictions for the task at hand. The second condition means that the real distribution of the target variable y_t has a similar amount of noise at each step.

If our task satisfies either one of these two conditions, then valuable information from the past could make the biggest contribution for the elements with the highest values of average "surprise" $H(y_t|x_t)$.

We should note that estimation of $I(y_j, m_j^*|x_j)$ by the local entropy makes MemUP vulnerable to the Noisy-TV problem[21], when the conditions we have described are not satisfied. Sensitivity tests for the Noisy-TV problem are discussed in Section 7.

Thus, instead of optimizing the memory for each future step, we introduce a new objective function that allows us to train it only for steps where the potential gain from the past information is maximal:

$$\max_{\theta} \sum_{k \in \mathcal{U}_t} I(y_k; m_t = g_\theta(x_t, m_{t-1})|x_k)\,, \tag{4}$$

here $\mathcal{U}_t$ denotes the set of top-K indices of steps from t to T with the highest estimated local entropy $H(y_k|x_k)$. The hyperparameter K controls fraction of a sequence to be processed in one gradient step. On the other hand, maximizing the mutual information between the memory m_t and an arbitrary distant event at step $k > t$ allows to learn long-term dependencies.

Optimization. Directly optimizing mutual information can be a challenging task. In our case, we use variational lower bound on mutual information proposed by [22]. Using the lower bound we can maximize mutual information by minimizing Cross-Entropy (CE) between the empirical and model distribution for all selected high entropy events:

$$\min_{\theta,\phi} \sum_{k \in \mathcal{U}_t} \mathbb{E}_{x_k, y_k} \left[-\log q_\phi(y_k|m_t^\theta, x_k) \right]\,, \tag{5}$$

where m_t^θ is a shortcut for $m_t = g_\theta(x_t, m_{t-1})$, q_ϕ is a predictor network with parameters ϕ, that estimates probability of y_k given x_k and m_t. For a detailed derivation of eq. 5 see Supplementary Materials Section A.

Collection of elements with high uncertainty $\mathcal{U}_t$ requires a special uncertainty detector d_ψ model. The main requirement for d_ψ is to produce uncertainty estimates $s_t = d_\psi(x_t)$. In the simplest case, the detector estimates "surprise" $-\log p(y_t|x_t)$ (prediction error), which can be seen as a single point estimate of uncertainty. Other models that directly estimate the uncertainty or a whole distribution [23; 24] can also be used.

3 MemUP for Recurrent Neural Architectures

Recurrent neural net implementation of MemUP consists of (1) training of uncertainty detector model d_ψ, (2) selection of elements with highest uncertainty $\mathcal{U}_t$ and (3) training memory g_θ and predictor q_ϕ models. A pseudocode for RNN MemUP training is shown in Supplementary Materials E.

Uncertainty detector training. Information maximization reasoning does not limit the choice of solutions for uncertainty detector d_ψ. In our experiments on algorithmic tasks we use a recurrent neural classifier trained with Cross-Entropy Loss. In this case detector's "surprise" $-\log\, d_\psi(y_t|x_t)$ is used as an uncertainty estimate.

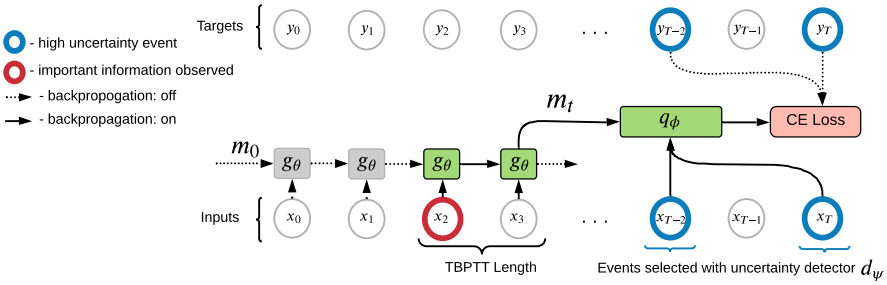


Figure 1: MemUP gradient update. The recurrent memory g_θ processes a sequence. The number of prediction targets $K = 2$ and TBPTT length $r = 2$. Blue circles denote states with the highest uncertainty estimates $d_\psi(y_k|x_k)$. Here x_2 marked by red circle contains information that can help to predict outcomes y_{T-2} and y_T. At the end of RNN TBPTT rollout these states are selected to form a set $U_3 = \{T-2, T\}$. Then, m_3, x_{T-2}, y_{T-2}, x_T, y_T are used to compute CE-Loss according to eq. 5. While states 2 and T could be separated by many millions steps, MemUP allows memory to capture the utility of information from x_2 by propagating gradients only through the sequence of $K + r = 4$ elements at a time.

In Reinforcement Learning experiments we use distributional algorithm QRDQN [24] trained to predict discounted future returns R_t ($y_t = R_t$ in RL experiments). As QRDQN approximates a whole distribution of the future returns, estimating uncertainty becomes a simple task.

Processing a sequence with the memory network. MemUP allows to learn long term-dependencies without propagating gradients through the whole sequence. To demonstrate that in our experiments we use Truncated BPTT to train memory network g_θ. In the majority of supervised learning experiments we set rollout length r between 10 and 60 steps. In all Reinforcement Learning experiments $r = 1$, i.e. recurrent memory is trained without actually using backpropagation through time. In All Experiments g_θ is a simple LSTM network with additional input encoder.

Selecting states with high uncertainty. Given uncertainty estimates $\{s_i\}_{i=t}^{T}$ for each future element in the sequence, we use softmax sampling $p(k) = e^{s_k/\tau} / \sum_{i=t}^{T}(e^{s_i/\tau})$ without replacement with $\tau = 0.02$. To implement sampling without replacement Gumbel-Max Trick [25] is used. A scheme illustrating a single gradient update is shown in Figure 1.

4 Related work

The problem of learning long-term dependencies has long been studied in both supervised learning [26; 27] and reinforcement learning domains [28]. However, most of the research related to recurrent networks architectures is aimed at solving the problems of exploding and vanishing gradients [2; 29; 7; 30]. The common trend is that many of the existing solutions to deal with vanishing gradients simultaneously increase the cost associated with the calculation of these gradients [29; 7]. For example, Transformers[10] generally require to store $O(N^2)$ of intermediate results for training on sequences of length N. The recently proposed Linformer [31] and BigBird [20] architectures allow training Transformers with linear complexity in space. However, most of these new Transformers have not yet found their way in the reinforcement learning setting.

In the field of deep reinforcement learning, the dominant approaches to implementation of memory are based on recurrent neural networks in combination with modern reinforcement learning algorithms [5; 32; 33; 34]. There are many studies that propose memory architectures for specific features of the reinforcement learning problems [9; 8]. In the paper by Parisotto and Salakhutdinov [35], a specifically modified DNC [7] architecture learned long-term dependencies of several hundred steps in the setting of 2D and 3D navigation tasks. The downside of this architecture is that the agent needs access to the information about its absolute or relative location in a 2D/3D environment. In another work, Ha and Schmidhuber [36] used a complex procedure with pre-training of memory and state embeddings. Their overall learning procedure is similar to ours. However, their memory/world model is trained to make predictions about the next step observations only, while we train memory to make predictions about events that can be arbitrarily far in the future.

Wayne at al. [17] demonstrated the state of the art results with the MERLIN algorithm. MERLIN's memory kept observation embeddings trained with a complex variational autoencoder in a DNC-

like [7] list. The memory module was also trained separately from the policy. Hung et al. [37] followed up on the MERLIN architecture's success and used the soft attention mechanism over all past embeddings to encourage exploratory actions in the POMDP environment. Their experiments showed models learning temporal dependencies with a length of 500+ steps. In contrast to our work, both of these solutions still require processing an arbitrarily long sequence of state embeddings by the policy networks at each step.

The direct attempts in applying Transformer architecture in reinforcement learning setting resulted in conclusion that Transformers are too unstable to work properly in RL [38]. Parisotto et al. [18] were able to overcome the instability of Transformers, showing state of the art result in the memory dependent 3D tasks. Another recent work, by Loynd et al. [39], has also applied Transformers to the POMDP problems, but the WMG architecture relies on availability of factored observations to the agent which imposes additional requirements on the environment. Switching focus from Transformers and complex architectures, Beck et al. [16] proposed AMRL architecture: a simple modification to the LSTM architecture to combat the problem of learning long-term dependencies in environments with a significant amount of observational noise.

5 Supervised Learning Experiments

For evaluation and comparison of our method we use four tasks: Copy [40], Scattered copy, Add [2] and permuted sequential MNIST (pMNIST) [41]. All these tasks are benchmarks that are used for testing models with long-term memory. In the original Copy task a sequence of length l ($l = 10$ in out experiments) should be copied after a $T - l$ steps ($T \in \{120, 520, 1020, 5020\}$ in our experiments), and in Scattered copy task a model has to make predictions in locations inside range $[l, T-1]$ that are chosen at random. We add this task in order to make detection of high-uncertainty locations harder. For Copy, Scattered copy and Add tasks train and test datasets have sizes 10K and 1K, for pMNIST 60K and 10K. For detailed tasks description see Section B.

In these experiments predictor q_ϕ consists of a three-layered MLP and a recurrent input encoder. The encoder E^{rnn} consists of a single fully-connected layer followed by two LSTM-layers with 128 hidden units and dropout probability 0.1. E^{rnn} is used to encode bigger local context than original inputs corresponding to indices in the U_t set (it is also possible to use a feedforward network for this task). Memory g_θ has the same architecture as E^{rnn} with separate weights.

We do not train a separate detector d_ψ. Uncertainty at each step is estimated by prediction error from the predictor module q_ϕ. For more information you can look at our implementation (see section E).

In the Add Task the model is trained with MSE loss, which is a special case of CE loss under the model-assumption that target distribution is a Gaussian with unit variance. Both memory and encoder are trained with the same Truncated BPTT length. Truncation length equals 10 in all tasks with sequence length < 1000, 20 in tasks with length ≥ 1000 and 30, 60 in pMNIST 784, 3136 respectively.

We compare MemUP with LSTM [2] and Transformer [10] models from pytorch library, as well as a recurrent network SRNN [30] which is designed for training on long sequences and has reduced saturation of gradients. LSTM* and SRNN* have the same architecture as E^{rnn} with 3 layered MLP on top. In the case of SRNN baselines, we have replaced all LSTM layers with SRNN layers.

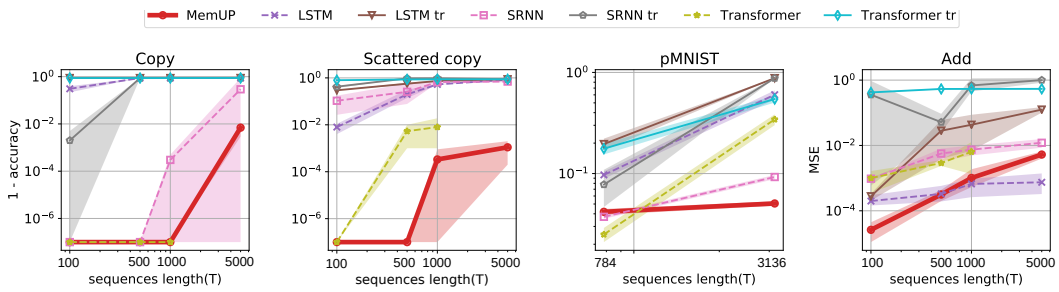


Figure 2: Final performance on supervised learning tasks. X-Axis shows task scaling, while Y-axis correspond to metric score at the end of training. Metrics: Inverted Accuracy (1. - Accuracy) in Copy, Scattered copy and pMNIST tasks, MSE in Add task. All curves are averaged over 3 runs.

The Transformer model consists of a single FC layer followed by 4 self-attention layers and the same 3 layered MLP on top. Baselines were trained in two modes. In the first mode (columns with Transformer, SRNN, LSTM) all elements of input sequence of length T were fed into a model simultaneously, so gradients propagate through the whole sequence during the training. We don't have results for Transformer on sequences of 5000+ steps since it didn't fit into our GPU RAM. In the second mode (Transformer Tr., SRNN Tr., LSTM Tr.) BPTT rollout for recurrent architectures and attention window for Transformer were truncated to process the same number of steps as MemUP ($r + K$).

Results in Figure 2 show that only MemUP allows to achieve a good quality of prediction in all tasks. In truncated BPTT setting (solid lines) MemUP is significantly better than LSTM tr, SRNN tr or Transformer tr due to MemUP ability to learn temporal dependencies longer that the length of BPTT rollout. When full information about the sequence is available to LSTM, SRNN and Transformer (dashed lines) but not MemUP, the latter still shows the best results for longest versions of Copy, Scattered Copy, pMnist, and has better overall performance.Moreover, when compared with Full BPTT baselines, MemUP requires to store much less intermediate computations (10-250 times) due to usage of short Truncated BPTT rollouts.

6 Reinforcement Learning Experiments

Performance of the MemUP algorithm is studied in two reinforcement learning tasks. The first task is classic T-Maze environment where the main difficulty is a long-term dependency between a hint at the starting position and location of a reward at the exit (see Figure 3.1). The main advantage of this problem is that it allows to test the agent's memory mechanisms in isolation because of primitive policy and the observation space. We base our experiments on the noisy version introduced by [16].

The second task is the color dependent object collection in the Doom environment [42; 43] which requires long-term memory in combination with a more complex policy and rich observation space. In the task introduced by [44], the agent is placed in a room filled with acid (see Figure 3.2.) and constantly loses health. Objects of green and red colors are scattered throughout the environment. Object of one color replenish the agent's health and give a +1 reward, while others take away health and give a -1 reward. The correspondence between effects and colors is determined randomly at the beginning of each episode. This information is conveyed by a column whose color matches the color of health replenishing items. In our version the column disappear after first 45 steps (see Figure 3.3). The episode ends when the agent's health drops to zero or after 1050 time steps.

The most straightforward extension of the MemUP algorithm to a reinforcement learning problem is to use memory state m_t as an additional input to an RL agent. There are several possible solutions to combine MemUp's training with agent's policy training: (1) pre train MemUP and then train an agent, (2) use alternating memory and agent training phases, train memory and agent in parallel, (3) combining the MemUP and RL agent into a single into an end-to-end network. In these experiments, we test the simplest version: train MemUP on trajectories generated with a frozen policy, then train an agent with a frozen MemUP memory. The second version is also implemented in the code (see Section E).

Inputs x_t are formed from observations o_t and previous action a_{t-1}. This time Observations are encoded using convolutional networks or fully connected networks. We use discounted future returns

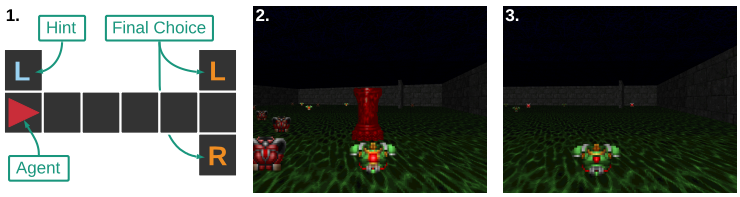
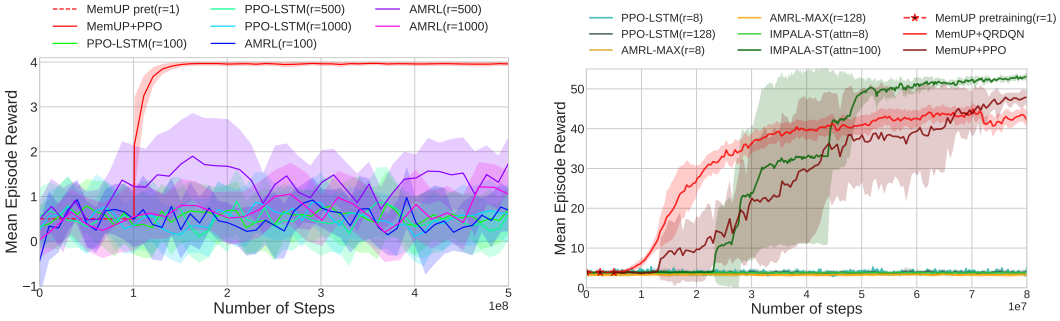


Figure 3: Environments for testing long-term time dependencies. **1.** *T-Maze environment.* The agent should reach the T-shaped junction and choose one of the arms (L or R). A hint about what arm to choose is provided at the very beginning. **2.** and **3.** *Vizdoom-Two-Colors environment*. The agent is in the room and constantly loses health. To replenish his health and recieve a reward the agent needs to collect items of the same color as the column. The color of the column is chosen randomly from red or green options at the beginning of the episode. After 45 steps from the beginning the column disappears. The episode lasts for a maximum of 1050 steps.



(a) (b)

Figure 4: **a)** Learning curves for MemUP and baselines in T-Maze-LNR-100. **b)** Learning plots in the Vizdoom-Two-Colors environment for all agents. All curves are averaged over 3 runs. Translucent areas around each curve show the standard deviation computed over all runs. The memory pre-training phase of MemUP agent is marked with the dashed line. TBPTT length/attention span for each baseline is specified in the parenthesis.

$R_t = \sum_{i=t}^{T} \gamma^{i-t} r_i$. as targets y_t. In the experiments, MemUP is compared with the following baselines:

PPO-LSTM is a recurrent version of PPO [45] with a single LSTM-layer [2]. We use PPO-LSTM implementation from the RLPyt library [46].

IMPALA-ST is an IMPALA [47] agent using Stabilized Transformer[10] architecture. Stabilized Transformer was presented by Parisotto et al. [18]. For experiments, we use the only available open implementation of this algorithm [48].

AMRL is proposed by [16] for the Noisy T-Maze Task. In the original study AMRL was compared with many different memory architectures including DNC, multi-layered LSTM and outperformed all of them on T-Maze-LN-100 task as well as other memory experiments. AMRL is similar to PPO-LSTM baseline, but extends LSTM with AMRL Layer.

In all experiments we use the same encoder networks for observation embeddings for all baselines and MemUP. Additionally the size of an LSTM-layer is the same for the memory module g_θ, PPO-LSTM and PPO-AMRL baselines. Results for the T-Maze-LNR-1000 (10 times longer dependency than in [16]) environment are presented in Figure 4a. The designation T-Maze-LNR-1000 means that the maze length (and corresponding tempral dependency) in each episode can be from 1000 to 1009 steps. Several rollout lengths are tested for RNN-based baselines. We could not successfully train a single PPO-LSTM or AMRL agent even with TBPTT rollout of $r = 1000$[1].

During MemUP pre-training phase, episodes are generated with a random policy. We use the discounted future reward with $\gamma = 0$ as a prediction target. A policy training for the MemUP agent starts after the memory pre-training phase is completed. The time spent in the pre-training phase is marked with a dashed line in all figures. As shown in Figure4a the agent learns almost instantly, given a memory pre-trained with our method. For the MemUP agent we train all components including memory module and uncertainty detector for each individual run from scratch. Additionally, Section D shows that MemUp can be trained to solve T-maze-LNR-20000.

Results for the Vizdoom-Two-Colors environment are shown in Figure 4b. IMPALA-ST(attn=8) with a short attention span could not learn any reasonable policy. On the other hand, IMPALA-ST(attn=100) solves the problem and reliably survive in the environment for 1050 steps. Both PPO-LSTM and AMRL baselines with TBPTT rollouts of 128 and 8 steps could not learn to survive longer than a random agent. While the rollout of 128 steps is potentially enough to remember the color of the column or a previously collected items, PPO-LSTM and AMRL agents are unable to utilize long-term information from the past.

For MemUP we use the discounted future reward with $\gamma = 0.8$ as a prediction target. The memory module g_θ is trained with the rollout length of 1 step. In Vizdoom-Two-Colors we set number of

[1] We tested our AMRL implementation on T-Maze-LNR-100 with full rollout and get results aligning with the original paper. But the algorithm did not withstand 10 times increase in episode length and truncation constants

targets $K = 3$. Thus, to train the memory g_θ we use 4 separate observations per episode for one gradient update. In ViZDoom, we train PPO (MemUP+PPO) and QRDQN (MemUP+QRDQN) agents with MemUP pretraining.

Both MemUP+PPO and MemUP+QRDQN learn to survive in the environement for full 1050 steps by collecting healing items. Though the final performance of IMPALA-ST(attn = 100) is slightly better. We speculate that this is due to the fact that the pre-trained memory **M** has learned to store mostly information about the color of the column, while IMPALA-ST (attn=100) has slightly better spatial awareness. On the other hand, MemUP use significantly fewer resources in terms of the size of processed sequences during training/pre-training and evaluation. As well, MemUP significantly outperforms baselines with comparable resource requirements, like IMPALA-ST (attn=8), PPO-LSTM(r=8) and AMRL-MAX(r=8).

7 Sensitivity to the Noisy-TV problem

In section 2 we discussed conditions on true uncertainty of targets y_t given a perfect memory m_t^*. The goal was to set conditions under which we can make a good guess about relative values of potential memory utility $I(y_t|m_t^*, c_t)$ by estimating local entropy $H(y_t|c_t)$. However, this may not be the case. For example, some targets y_t may be completely unpredictable and no information from the past can reduce the entropy for the distribution of y_t: $H(y_t|x_t) \sim H(y_t|m_t^*, x_t)$. Then the local entropy is large, while potential utility of memory for y_t predictions is close to zero. Training memory to predict such targets is pointless and prevents MemUP from focusing on events for which training long-term memory is essential. The problem of over-emphasis on unpredictable surprising events is called Noisy-TV problem[21]. Noisy-TV problem is often encountered by algorithms using curiocity-based exploration in reinforcement learning[49]. To test robustness of MemUp to the Noisy-TV problem, we have modified T-maze-LNR-100 environment.

Noisy T-maze With Distractors. In this version, agent can receive +4 or -3 rewards in D+1 decision points. Their location in the corridor is chosen randomly at a distance of at least 50 steps from the hint. All of them can be detected when $o_t[1]! = 0$. element in observation vector o_t. Decision points are distinguishable from each other. Each decision point has a unique value of $o_t[1]$. Only in one decision point, the next reward depends on the agent's action and the value of the hint ($o_t[1] = 1$. in this case). In other D decision points, the next rewards are completely random (chosen with a probability of 0.5). Thus, Noisy T-maze With Distractors generate sequences with one long-term dependency and D events acting as Noisy-TV distractors.

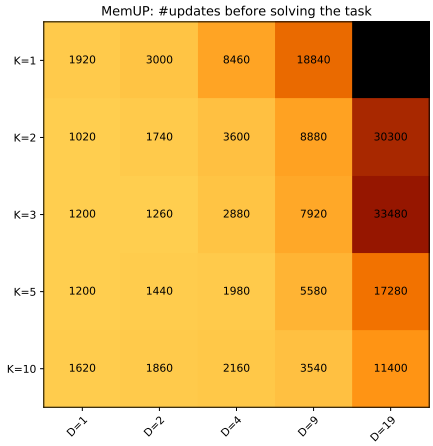


Figure 5: Each cell shows the mean number of updates required for the memory to correctly predict (achieving 0.1 RMSE) the reward after the decision point connected with the temporal dependency. The x-axis shows the number of unpredictable decision points in the environment. The Y-axis shows number of events predicted by MemUP. Darker colors means slower solution.

We conducted 25 experiments with 6 independent runs in each (150 runs in total). The differences in the experiments come down to two parameters: number of distracting events D (X-axis in Fig. 5) and number of events sampled for prediction K (Y-axis in Fig. 5). More distracting events make the task of learning the temporal dependency harder. Increasing the size of $\mathcal{U}_t$ set(by changing K) improves the chance of selecting a future event that is a part of temporal dependency. On the other hand, with an increase in $\mathcal{U}_t$, the resource cost for memory training also increase. The TBPTT rollout length in these experiments is 1.

Results in Fig. 5 show that with an increase in the number of distractions, memory learning speed decreases. However, increasing the set $\mathcal{U}_t$ allows to alleviate this problem. For the experiment (D=19, K=1), none of the runs solved the task in the first 45000 updates. In experiments (D=19, K=2) and (D=19, K=3) only 4 out of 5 runs solved the task. The results show that the agent can learn the time dependency fairly quickly even if the uncertainty detector select noisy events 80% of the time.

8 Ablation study

To study an effectiveness of our core idea, i.e. training memory by predicting long-term future events with high uncertainty, we compared MemUP with baselines that exclude core MemUP features while sharing the same neural architecture.

In these experiments, we carefully consider the process of learning memory (without further training the RL agent) on the T-Maze-LNR problems. The following memory ablation baselines were studied:

MemUP: Proposed in this paper. See section 2.

Rnd-Pred: Same as MemUP, but steps for future outcome prediction are selected randomly and uniformly among all future timesteps. This randomization verify whether predicting steps with high uncertainty helps MemUP to overcome the problem of learning long-term dependencies with small BPTT rollouts.

Default: Same as MemUP, but it is trained to predict return R_t at each step t, as oposed to an arbitrarily distant future events as in MemUP and *Rnd-Pred*. Thus, *Default* is the same as the LSTM baseline in the Supervised Learing Experiments (see 5). But in this study we test this baseline on all intermediate Truncated BPTT lengths.

All versions have the same architecture, and hyperparametes. They were trained in T-Maze-LNR-100 and T-Maze-LNR-1000 problems. Episodes were generated by a random strategy. All versions are trained with 5 different Truncated BPTT lengths: 1, 5, 10, 50, 100. Testing metric is a root-mean-squared error(RMSE) between the models' prediction and the actual return at the final step of the episode. The evaluation was conducted on the 100 separately recorded episodes.

Figure 6 shows the results of the experiments. On the x-axis, we have specified the length of the Truncated BPTT in decreasing order, i.e. in the order of increasing task complexity. For every rollout length there is a corresponding absolute value of error between predictions of the model and targets

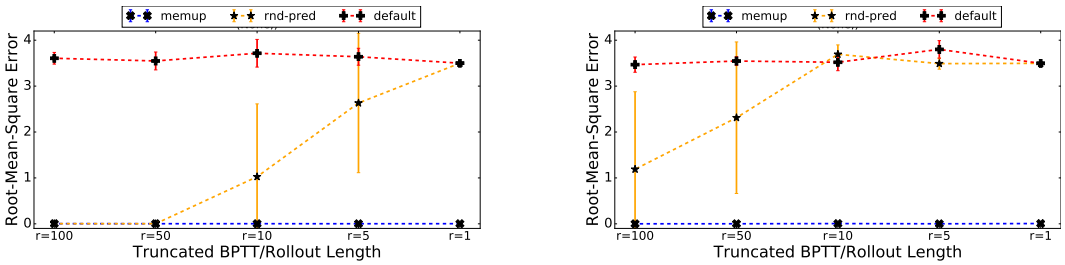


Figure 6: Averaged over 6 runs prediction error at the end of training for each ablation baseline with respective TBPTT length. Vertical bars show standard deviation computed over 6 runs. **a)** T-Maze-LNR-100. **b)** T-Maze-LNR-1000.

at the end of training ($3e4$ updates for T-Maze-LNR-100 and $3e5$ for T-Maze-LNR-1000). Smaller values represent better predictions.

The *Default* modification is unable to learn a long-time dependency exceeding the length of the tested rollouts. This is the same as the results of the PPO-LSTM agent on the T-Maze-LNR Reinforcement Learining experiments.

The *Rnd-Pred* modification solves the task with rollouts 100 and 50 in T-Maze of length 100, but for shorter rollouts and for longer T-Maze-LNR-1000 results deteriorate significantly. In the majority of runs in T-Maze-LNR-1000, the memory has not been learned. The relative success of *Rnd-Pred* is expected because random selection of prediction target still allows to sample a useful future event with some small probability.[2] Thus, random sampling of prediction targets allows to learn casual dependency that is 2 times longer than TBPTT, but already struggles at ratio of 10.

On the other hand, the MemUP is able to solve the problem regardless of the rollout length, i.e. MemUP demonstrated an ability to learn casual dependencies that 1000 times longer than its TBPTT length in this setting. In this study MemUP is successfully trained with TBPTT of length 1, in other words, without using backpropagation through time at all.

9 Conclusion

In this paper, we proposed a new method for training long-term memory. The main idea is to train a memory network to predict future outcomes of high uncertainty and skip all others. Predicting a small number of arbitrarily distant future outcomes substantially saves computational resources required to backpropagate gradients. At the same time, the emphasis on high uncertainty outcomes allows not to miss long-term dependencies in the task.

Experimental results show that our training algorithm allows to learn temporal dependencies significantly longer than the number of steps processed for a single gradient update. None of the baseline architectures trained in a classical way demonstrate such ability. All these baselines with comparable results require to store at least 200 times more intermediate calculations for the Add, Copy, Scattered Copy, pMNIST tasks as well as at least 500 times more for the T-maze, and 625 times more for the Vizdoom-Two-Colors (Stabilized Transformer baseline has quadratic complexity while processing 25 times more steps).

Even using fewer resources, MemUP outperforms non-truncated baselines that has simultaneous access to all elements of a sequence (including Transformer) on pMNIST(3136), Copy and Scattered Copy tasks. On T-maze and Vizdoom-Two-Colors tasks, MemUP is better than all recurrent baselines. Another advantage is that MemUP can be combined with any recurrent architecture and applied both to supervised as well reinforcement learning settings.

We believe that the MemUP algorithm applied to recurrent networks has shown promising results, and the main idea demonstrates exciting avenues for future research.

References

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

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

[3] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv:1410.5401*, 2014.

[4] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv:1410.3916*, 2014.

[5] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[2] For example, for a rollout of 50, the chance that at least one of the memory steps will need to predict the correct target event is close to 0.5 in T-Maze-LNR-100.

[6] Junhyuk Oh, Valliappa Chockalingam, Satinder P. Singh, and Honglak Lee. Control of memory, active perception, and action in minecraft. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2790–2799. JMLR.org, 2016. URL `http://proceedings.mlr.press/v48/oh16.html`.

[7] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538 (7626):471, 2016.

[8] Adam Santoro, Ryan Faulkner, David Raposo, Jack W. Rae, Mike Chrzanowski, Theophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy P. Lillicrap. Relational recurrent neural networks. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 7310–7321, 2018. URL `https://proceedings.neurips.cc/paper/2018/hash/e2eabaf96372e20a9e3d4b5f83723a61-Abstract.html`.

[9] Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. *7th International Conference on Learning Representations (ICLR)*, 2019.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 5998–6008, 2017. URL `http://papers.nips.cc/paper/7181-attention-is-all-you-need`.

[11] Mikhail S Burtsev, Yuri Kuratov, Anton Peganov, and Grigory V Sapunov. Memory transformer. *arXiv preprint arXiv:2006.11527*, 2020.

[12] Jie Lei, Liwei Wang, Yelong Shen, Dong Yu, Tamara Berg, and Mohit Bansal. MART: Memory-augmented recurrent transformer for coherent video paragraph captioning. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2603–2614, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.233. URL `https://aclanthology.org/2020.acl-main.233`.

[13] Ankit Gupta and Jonathan Berant. Gmat: Global memory augmentation for transformers. *arXiv preprint arXiv:2006.03274*, 2020.

[14] Qingyang Wu, Zhenzhong Lan, Jing Gu, and Zhou Yu. Memformer: The memory-augmented transformer. *arXiv preprint arXiv:2010.06891*, 2020.

[15] Angela Fan, Thibaut Lavril, Edouard Grave, Armand Joulin, and Sainbayar Sukhbaatar. Addressing some limitations of transformers with feedback memory. *arXiv preprint arXiv:2002.09402*, 2020.

[16] Jacob Beck, Kamil Ciosek, Sam Devlin, Sebastian Tschiatschek, Cheng Zhang, and Katja Hofmann. AMRL: aggregated memory for reinforcement learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=Bkl7bREtDr`.

[17] Greg Wayne, Chia-Chun Hung, David Amos, Mehdi Mirza, Arun Ahuja, Agnieszka Grabska-Barwinska, Jack W. Rae, Piotr Mirowski, Joel Z. Leibo, Adam Santoro, Mevlana Gemici, Malcolm Reynolds, Tim Harley, Josh Abramson, Shakir Mohamed, Danilo Jimenez Rezende, David Saxton, Adam Cain, Chloe Hillier, David Silver, Koray Kavukcuoglu, Matthew Botvinick, Demis Hassabis, and Timothy P. Lillicrap. Unsupervised predictive memory in a goal-directed agent. *CoRR*, abs/1803.10760, 2018. URL `http://arxiv.org/abs/1803.10760`.

[18] Emilio Parisotto, H. Francis Song, Jack W. Rae, Razvan Pascanu, Çaglar Gülçehre, Siddhant M. Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, Matthew M. Botvinick, Nicolas Heess, and Raia Hadsell. Stabilizing transformers for reinforcement learning. *CoRR*, abs/1910.06764, 2019. URL `http://arxiv.org/abs/1910.06764`.

[19] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[20] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/c8512d142a2d849725f31a9a7a361ab9-Abstract.html`.

[21] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

[22] David Barber Felix Agakov. The im algorithm: a variational approach to information maximization. *Advances in neural information processing systems*, 16(320):201, 2004.

[23] Evgenii Tsymbalov, K. Fedyanin, and Maxim Panov. Dropout strikes back: Improved uncertainty estimation via diversity sampled implicit ensembles. *ArXiv*, abs/2003.03274, 2020.

[24] Will Dabney, Mark Rowland, Marc G Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[25] Iris AM Huijben, Wouter Kool, Max B Paulus, and Ruud JG van Sloun. A review of the gumbel-max trick and its extensions for discrete stochasticity in machine learning. *arXiv preprint arXiv:2110.01515*, 2021.

[26] Herbert Jaeger. *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*, volume 5. GMD-Forschungszentrum Informationstechnik Bonn, 2002.

[27] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. *A field guide to dynamical recurrent neural networks. IEEE Press*, 2001.

[28] Bram Bakker. Reinforcement learning with long short-term memory. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pages 1475–1482. MIT Press, 2001.

[29] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014. URL `http://arxiv.org/abs/1410.5401`.

[30] Michael Rotman and Lior Wolf. Shuffling recurrent neural networks. *CoRR*, abs/2007.07324, 2020. URL `https://arxiv.org/abs/2007.07324`.

[31] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *CoRR*, abs/2006.04768, 2020. URL `https://arxiv.org/abs/2006.04768`.

[32] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *CoRR, abs/1507.06527*, 7(1), 2015.

[33] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 3540–3549. PMLR, 2017. URL `http://proceedings.mlr.press/v70/vezhnevets17a.html`.

[34] Bowen Baker, Ingmar Kanitscheider, Todor M. Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=SkxpxJBKwS`.

[35] Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL `https://openreview.net/forum?id=Bk9zbyZCZ`.

[36] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 2455–2467, 2018. URL `https://proceedings.neurips.cc/paper/2018/hash/2de5d16682c3c35007e4e92982f1a2ba-Abstract.html`.

[37] Chia-Chun Hung, Timothy Lillicrap, Josh Abramson, Yan Wu, Mehdi Mirza, Federico Carnevale, Arun Ahuja, and Greg Wayne. Optimizing agent behavior over long time scales by transporting value. *Nature communications*, 10(1):1–12, 2019.

[38] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL `https://openreview.net/forum?id=B1DmUzWAW`.

[39] Ricky Loynd, Roland Fernandez, Asli Çelikyilmaz, Adith Swaminathan, and Matthew J. Hausknecht. Working memory graphs. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 6404–6414. PMLR, 2020. URL `http://proceedings.mlr.press/v119/loynd20a.html`.

[40] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. *International Conference on Machine Learning*, pages 1120–1128, 2016.

[41] Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *ArXiv*, abs/1504.00941, 2015.

[42] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2016.

[43] Guillaume Lample and Devendra Singh Chaplot. Playing FPS games with deep reinforcement learning. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 2140–2146. AAAI Press, 2017. URL `http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14456`.

[44] Edward Beeching, Christian Wolf, Jilles Dibangoye, and Olivier Simonin. Deep reinforcement learning on a budget: 3d control and reasoning without a supercomputer. *CoRR*, abs/1904.01806, 2019. URL `http://arxiv.org/abs/1904.01806`.

[45] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[46] Adam Stooke and Pieter Abbeel. rlpyt: A research code base for deep reinforcement learning in pytorch. *arXiv preprint arXiv:1909.01500*, 2019.

[47] Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1406–1415. PMLR, 2018. URL `http://proceedings.mlr.press/v80/espeholt18a.html`.

[48] Shakti Kumar, Jerrod Parker, and Panteha Naderian. Adaptive transformers in rl. *arXiv preprint arXiv:2004.03761*, 2020.

[49] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] Our experimental results and ablation study align with our claims. See Sections 6 and 8.
   (b) Did you describe the limitations of your work? [Yes] See Section 2 and Noisy-TV experiments 7
   (c) Did you discuss any potential negative societal impacts of your work? [No] No, Negative consequences of practical application of our idea (that we could think of), can be coupled with any improvement in DL/RL methods rather than specifically with our proposal.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] We describe our assumptions in Section 2 (L.83-96)
   (b) Did you include complete proofs of all theoretical results? [Yes] See Section A.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Link to the code is given in Section E
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We have included these details in *Supplementary Materials*
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] All graphs include mean and standard deviation computed over multiple runs with different random seed.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See *Supplementary Materials*.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We use RLPyt library and open implementation of Stabilized Transformer and SRNN for our baselines and cite authors in sections 5 and 6
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No]

A Mutual Information Maximization

Estimating mutual information $I(y_k, m_t|x_k)$ can be difficult task from a computational point of view. Therefore, to make the task more trackable we use a lower bound for mutual information derived by [22]. Adapting it for the MemUP case, we get the following inequality:

$$I(y_k|m_t^\theta, x_k) = H(y_k|x_k) - H(y_k|m_t^\theta, x_k) \geq H(y_k|x_k) - CE(p(y_k|m_t^\theta, x_k), q_\phi(y_k|m_t^\theta, x_k)), \tag{6}$$

where m_t^θ is a shortcut notation denoting dependence of the memory state m_t on the parameters of memory network g_θ. The distribution p is the true conditional distribution of future outcomes y_k, which is unknown to us, but we can sample from it using training data. The distribution q_ϕ is approximated by the predictor network with parameters ϕ. Inequality 6 follows from the relation between KL-divergence, cross-entropy (CE) and entropy: $D_{KL}(p||q_{phi}) = CE(p, q_\phi) - H(p)$. Using non-negativity property of KL-divergence leads to $CE(p, q) \geq H(p)$. The bound becomes exact when $q_\phi(y_k|m_t^\theta, x_k)$ is equal to $p(y_k|m_t^\theta, x_k)$

Since $H(y_k|x_k)$ is independent from memory and predictor networks, maximizing the lower bound is the same as minimizing cross-entropy. Therefore, the memory g_θ and the predictor q_ϕ can be jointly trained to maximize mutual information by simply minimizing cross entropy loss:

$$\min_{\theta,\phi} \sum_{k \in \mathcal{U}_t}^{T} CE(p(y_k|m_t^\theta, x_k), q_\phi(y_k|m_t^\theta, x_k)) = min_{\theta,\phi} \sum_{k \in \mathcal{U}_t}^{T} \mathbb{E}_{x_k, y_k}[-log\, q_\phi(y_k|m_t^\theta, x_k)]. \tag{7}$$

Specifically, the memory module g_θ is trained by backpropagating gradients directly through the predictor network q_ϕ.

B Supervised Learning Tasks

For accuracy evaluation we involve four tasks: Copy, Scattered copy, Add and permuted MNIST (pMNIST). All these tasks are benchmarks that are used for testing models with long-term memory. Describe each task in details.

Copy task. This is an aligned sequence to sequence classification problem ($X \to Y$). Denote by T length of X and Y. The first 10 elements contain random uniform digits from set $\{2, \ldots, 9\}$ that model has to copy and reproduce in the end of the sequence where $X_i = 1, i \in \{T-10, \ldots, T-1\}$. In all other positions except the first 10 and the last 10 $X_i = 1, i \in \{10, \ldots, T-11\}$. Correspondingly $Y_i = 0, i \in \{0, \ldots, T-11\}$ except the last 10 where $Y_{T-10+i} = X_i, i \in \{0, \ldots, 9\}$. We split this task into four different sub-tasks by the length where $T \in \{120, 220, 520, 1020, 5020\}$. Train dataset contains 10k sequences and test consists of 1k sequences.

Scattered copy task. It is similar to the previous one. The only difference is that locations where one has to make predictions are choosen at random in range $[10, \ldots, T-1]$. In such locations $X_i = 1$ and Y_i equals to some element from the first 10 digits. Analogically we split this task into four different sub-tasks by the length where $T \in \{120, 220, 520, 1020, 5020\}$. This task is more complex because it requires locations detection and their ordered number counting. As a metric in the previous two tasks we use negative log-likelihood (NLL). Train dataset contains 10k sequences and test consists of 1k sequences.

Add task. This is an aligned sequence to sequence regression problem ($X \to Y$). Each element in X is two dimensional. The first dimension contains a random uniform value $\in [0, 1]$ and second component is 0 or 1. There are only three ones per sequence that correspond to two summands and their sum. Values of Y elements equal to zero except one element at the sum location. MSE metric was used here as a loss function and quality measure. Train dataset contains 10k sequences and test consists of 1k sequences. We split this task into four different sub-tasks by the length where $T \in \{100, 200, 500, 1000, 5000\}$.

Permuted sequential MNIST task (pMNIST). The dataset is obtained from the ordinary MNIST that includes 60k train images and 10k test images. Each image is flattened to one dimensional vector. A random permutation changes order of the vector elements consistently. In training procedure we apply NLL loss function and measure quality by classification accuracy in %.

Table 1: Supervised Learning Results. **Metrics:** Accuracy (classification accuracy in %) in Copy, Scattered copy and pMNIST tasks, MSE in Add task. **Bold** font highlights the best score per task, * shows the best score among truncated methods.

Task/Method	MemUP	LSTM	LSTM tr	SRNN	SRNN tr	Transf.	Transf. tr
Copy 120	**100***	69.3	12.7	**100**	99.7	**100**	12.5
Copy 520	**100***	12.4	13.0	**100**	13.1	**100**	12.7
Copy 1020	**100***	12.4	12.9	99.9	12.5	**100**	12.6
Copy 5020	**99.3***	12.4	12.5	70.8	12.5	out of mem	12.5
Scat. copy 120	**100***	99.2	70.4	89.4	57.8	**100**	20.3
Scat. copy 520	**100***	80.1	43.4	74.5	5.0	99.5	14.3
Scat. copy 1020	**99.9***	47.2	26.2	33.4	5.1	99.3	12.9
Scat. copy 5020	**99.9***	16.8	12.9	30.3	12.5	out of mem	12.9
Add 100	**0.00003***	0.00019	0.00027	0.00095	0.356	0.00103	0.420
Add 500	**0.00031***	0.00032	0.02830	0.00565	0.516	0.00291	0.536
Add 1000	0.00101*	**0.00066**	0.04294	0.00744	0.685	0.00638	0.537
Add 5000	0.00526*	**0.00074**	0.12550	0.01206	1.000	out of mem	0.546
pMNIST 784	95.4*	89.5	79.85	96.43	95.33	**97.1**	84.55
pMNIST 3136	**94.3***	33.6	11.7	90.31	11.7	63.5	49.7

Table 2: Memory size impact. We compute for MemUP method the dependence of MSE metric on LSTM layer's dimension in tasks Add 500 and pMNIST 784.

Task/Memory size	128	256	512	1024
Add 500	0.00028	0.00018	0.00024	0.00056
pMNIST 784	89.4	92.5	95.4	96.1

C Reinforcement Learning Tasks

Noisy T-Maze. Namely, the agent starts at the very beginning of the central corridor next to the hint that it observes in the first step of the episode. The agent can only move forward along the corridor. At each step, the agent's observations o_t are represented by a vector of length 3. The first element $o_t[0]$ contains value of the hint: +1 or -1 at the first step, 0 otherwise. The second element $o_t[1]$ is the indicator of reaching the intersection, which is 1 if the agent has reached the place of turn. The last element $o_t[2]$ does not carry information and is a random noise, it is +1 or -1 with equal probability. The agent receives the reward only at the end of the episode. It is +4 if the correct turn was chosen and -3 otherwise.

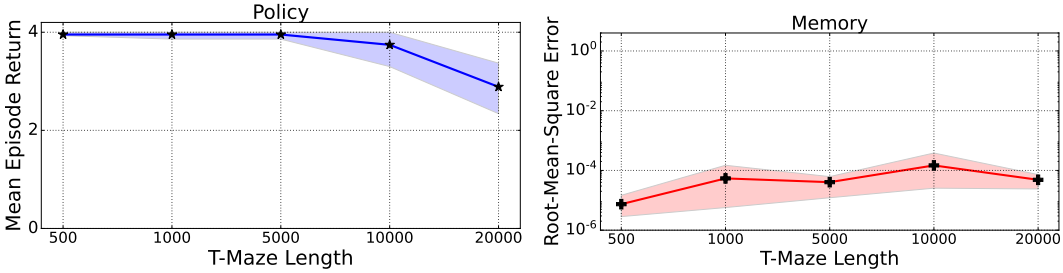
The length of the maze is defined as the number of steps between the moment the agent sees the hint and the moment the agent has to make a choice based on the value of the hint. The real maze length in experiments varies within 10 steps from episode to episode in order to decorrelate the observations of agents trained in parallel on multiple instances of environments (see PPO algorithms [45], A3C [5], IMPALA [47]). The naming of the environment indicates the minimum possible maze length, for example, the designation T-Maze-LNR-100 means that the maze length in each episode can be from 100 to 109 steps.

Vizdoom-Two-Colors. Despite the fact that T-Maze environment allows simulating very long temporal dependencies, it is too simplistic. To explore if MemUP augmentation can be scaled to much more complex tasks and environments it was tested in the Vizdoom-Two-Colors task introduced by Beeching et al. [44]. In this task, the agent is placed in a room filled with acid (see Figure 3.2.) and constantly loses health. Objects of green and red colors are scattered throughout the environment. Items of one color replenish the agent's health and give a +1 reward, while others take away health and give a -1 reward. The correspondence between effects and items' colors is determined randomly at the beginning of each episode. This information is conveyed by a column whose color matches the color of health replenishing items. The episode ends when the agent's health drops to zero or after 1050 time steps. At each step agent receives a small "living reward" equal to +0.02. Accordingly, the goal of the agent is to survive as long as possible in the environment by collecting items of the rewarding color.

To solve this problem, it is necessary to keep the color of the signal column in memory in order to be able to select objects of the correct color, even when the column is out of sight. However, in the course of preliminary experiments, it turned out that a reactive agent without memory is able to learn a strategy in which it will always keep the column in sight, even if the room is filled with walls blocking the view from many angles. Therefore, we created a new version of the environment, where the column disappears after the 45th step (see Figure 3.3.) and the number of walls in the room is significantly reduced. Thus, the subtask of memorization became harder in comparison with the original version, and the subtask of navigation in the environment was simplified. It is also worth noting that the agent does not receive information about the current health or the rewards received, since these observations actually provide the same information as the color of the column.

D T-Maze Scaling Experiments.

In this section, we test MemUP's scaling ability in the T-maze-LNR environment. For each version of the environment (with lengths 500, 1000, 5000, 10000, 20000) we train PPO+MemUP from scratch in 3 separate runs. Picture 7a shows average episodic return after 500 training steps of PPO in the respective environment. Picture 7b shows the root mean squared error of the MemUP memory and predictor modules after 200 training epochs. The evaluation of the memory is conducted in the same way as in Section 8. The memory module is trained with BPTT rollout $r = 20$ in all runs.



(a) (b)

Figure 7: **T-Maze Scaling Experiments. a)** Final PPO+MemUP performance with respect to corridor length in Noisy T-Maze. Each point is averaged over 3 runs. **b)** Final memory and predictor quality with respect to corridor length in Noisy T-Maze. Each point is averaged over 3 runs.

E Open Source Implementation

Implementation for MemUP can be found at: `https://github.com/griver/memup`

F Space Complexity for MemUP

Time complexity for MemUP would be the same as for RNNs trained with TBPTT. In our method we focus on space complexity, as MemUP allows RNNs to learn long dependencies with short rollouts.

Space complexity for MemUP is $O(k\frac{r}{f} + r)$, where:

- r - rollout length;
- f - prediction frequency (you can predict from any intermediate step inside a single rollout);
- k - number of prediction targets.

In all experiments (excluding Section 8) frequency (long-term predictions are made at the end of each rollout), therefore the space complexity is only $O(k + r)$. We would like to note that we recommend using $f < r$ in longer rollouts as it saves from the vanishing gradient problem inside a single rollout.

G Pseudocode for Memory Training

This section provides a simplified version of the memory training algorithm with feedforward predictor q_ϕ.

Algorithm 1 Recurrent Memory Training

Input: inputs x, targets y, r, K, d_ψ, q_ϕ, g_θ
Train detector d_ψ on the sequence data x and y
Get uncertainty estimates s for each step using d_ψ
$t = 0$ and $m = None$
while $t \leq T$ **do**
  for $i = t$ **to** $t + r$ **do**
    $m = g_\theta(x_i, m)$
  end for
  $t = \min(T, t + r)$
  Using $s[t : T]$ sample a set $\mathcal{U}_t$ of K elements with highest uncertainty
  Optimize CE loss from eq. 5: $\min_{\theta,\phi} \sum_{k \in \mathcal{U}_t} [-\log q_\phi(y_k | m, x_k)]$
  stop_gradient(m)
end while