Sparse Attentive Backtracking

Temporal Credit Assignment Through Reminding

Paper authors: Nan Rosemary Ke, Anirudh Goyal, Olexa Bilaniuk,
Jonathan Binas, Michael C. Mozer, Chris Pal, Yoshua Bengio

arXiv:1809.03702v1 (11 September 2018)

Published at NeurIPS 2018 | Official code

Central idea

Retrieve a few useful past states, then send credit directly to them and to short segments of their local history.

Independent reading presentation, 29 October 2026. Publication and code checked on this date.

Long memory also needs a route for learning

$$\boldsymbol{h}^{(t)} = f_\theta(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)})$$

Full BPTT

Credit crosses the entire chain of recurrent transitions.

Long products of Jacobians; a long computation history to retain.

Truncated BPTT

Credit stops after a short local horizon.

Forward state can retain information, while distant computations lose their direct training signal.

SAB: keep short local updates and add learned temporal shortcuts.

Paper, Secs. 1.1–1.2, PDF pp. 2–3.

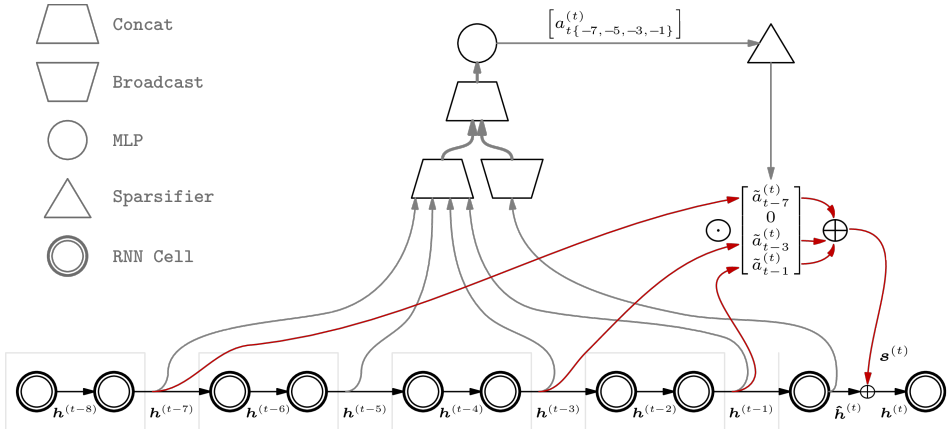
Three budgets control storage, reading and replay

Parameter	Meaning	Consequence
k_{att}	Save every k_{att}-th hidden state	Controls candidate memory density
k_{top}	Read at most k_{top} memories per step	Controls sparse retrieval support
k_{trunc}	Backpropagate through a short local segment	Controls local credit assignment

- Retrieval runs at every step; k_{att} is the *write* interval.
- A retrieved state can be much older than k_{trunc} time steps.
- Memory grows with history; this implementation has no eviction rule.

Paper, Secs. 3.1–3.2 and Algorithm 1, PDF pp. 4–5.

The forward pass selects a few temporal shortcuts



Gray: score all stored candidates. Red: read the selected memories.

Original Fig. 1, PDF p. 4; $k_{\text{top}} = 3$, $k_{\text{att}} = 2$, $k_{\text{trunc}} = 2$. Author image.

Current context scores every stored memory

$$(\hat{\boldsymbol{h}}^{(t)}, \boldsymbol{c}^{(t)}) = \mathrm{LSTMCell}(\boldsymbol{h}^{(t-1)}, \boldsymbol{c}^{(t-1)}, \boldsymbol{x}^{(t)})$$

$$\boldsymbol{d}_i^{(t)} = \boldsymbol{W}_1 \boldsymbol{m}^{(i)} + \boldsymbol{W}_2 \hat{\boldsymbol{h}}^{(t)}, \qquad a_i^{(t)} = \boldsymbol{W}_3 \tanh(\boldsymbol{d}_i^{(t)})$$

- $\hat{\boldsymbol{h}}^{(t)}$: provisional hidden state, used as the query.
- $\boldsymbol{m}^{(i)} \in \mathbb{R}^n$: a stored past hidden state.
- $a_i^{(t)}$: scalar relevance score from a one-hidden-layer MLP.

Where the work still happens

Sparse reading follows *all-candidate scoring*.

Paper, Sec. 3.1 and Algorithm 1, PDF pp. 4–5.

A threshold turns scores into sparse weights

Let τ_t be the $(k_{\text{top}}+1)$-th largest score.

$$\tilde{a}_i^{(t)} = \max\{a_i^{(t)} - \tau_t, 0\}, \qquad \alpha_i^{(t)} = \frac{\tilde{a}_i^{(t)}}{\sum_j \tilde{a}_j^{(t)}}$$

Teaching example: $k_{\text{top}} = 2$

$$[4,3,1,0] \xrightarrow{\tau=1} [3,2,0,0] \xrightarrow{\text{normalize}} [0.6,0.4,0,0]$$

- Positive weights measure prominence above the cutoff.
- At most k_{top} active memories, when normalization is defined.
- Unlike softmax, excluded memories have exactly zero read weight.

Algorithm 1, PDF p. 5. α is added notation for the normalized weight; example is illustrative.

Retrieved content updates both state and output

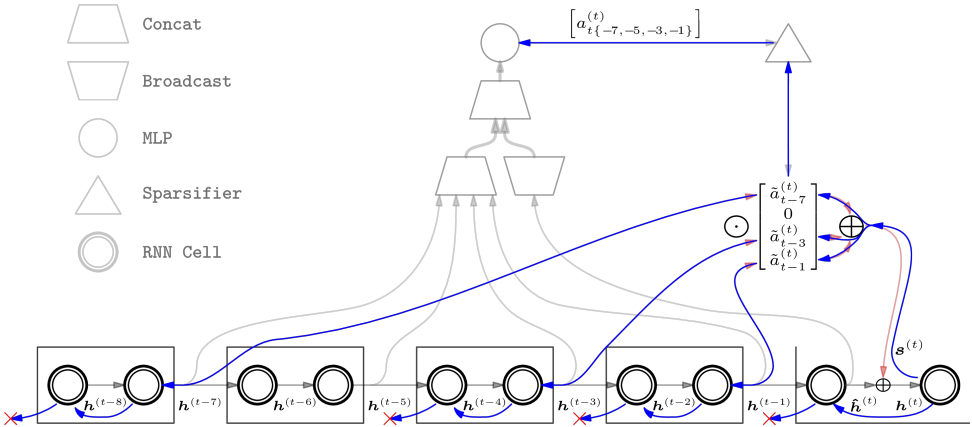
$$\boldsymbol{s}^{(t)} = \sum_i \alpha_i^{(t)} \boldsymbol{m}^{(i)}, \qquad \boldsymbol{h}^{(t)} = \hat{\boldsymbol{h}}^{(t)} + \boldsymbol{s}^{(t)}$$

$$\boldsymbol{y}^{(t)} = \boldsymbol{V}_1 \boldsymbol{h}^{(t)} + \boldsymbol{V}_2 \boldsymbol{s}^{(t)} + \boldsymbol{b}$$

1. Read past memories into the summary $\boldsymbol{s}^{(t)}$.
2. Add the summary to the provisional hidden state.
3. Predict using both the resulting state and the summary.
4. If $t \equiv 0 \pmod{k_{\text{att}}}$, append $\boldsymbol{h}^{(t)}$ to memory.

Paper, Algorithm 1, PDF p. 5. Retrieval occurs before the current state is appended.

Backward credit jumps, then replays a local segment



Blue: gradient flow. Red crosses: local truncation boundaries.

Original Fig. 2, PDF p. 6; $k_{\text{top}} = 3$, $k_{\text{att}} = 2$, $k_{\text{trunc}} = 2$. Author image.

Mental updates train the computation behind a memory

Direct value path, holding attention weights fixed

$$\left.\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{m}^{(i)}}\right|_{\text{direct, } \alpha \text{ fixed}} = \alpha_i^{(t)} \boldsymbol{I}$$

- The shortcut avoids a Jacobian product over the entire time gap.
- Mental updates extend credit to k_{trunc} local steps before a selected state.
- Truncation still gives an approximate gradient; score derivatives add other terms.
- Inference uses retrieval and state updates. Mental updates are a training operation.

Paper, Secs. 3.2 and 8.1, PDF pp. 5, 13. Displayed derivative is a teaching derivation.

The experiments separate memory, transfer and mechanism

Test	What must the model do?	Metric
Copying	Recall 10 initial symbols after a delay	Accuracy ↑
Adding	Sum two masked inputs	Loss ↓
PTB / Text8	Predict the next character	BPC ↓
Image sequences	Classify pMNIST / CIFAR10	Accuracy ↑
Length transfer	Train at $T = 100$; test longer delays	Accuracy ↑
Ablations	Remove sparsity or local replay	Task scores

Baselines: full-BPTT LSTM, TBPTT LSTM, full-attention LSTM;
Transformer results on the image-sequence tasks.

Paper, Sec. 4, PDF pp. 5–8; experimental details in Sec. 7, PDF p. 12.

Five-step local updates can learn a 300-step delay

Last-10-symbol accuracy (%); each column is trained at its own delay.

Method	$T = 100$	$T = 200$	$T = 300$
LSTM, full BPTT	99.8	56.0	35.9
LSTM + full attention, full BPTT	100.0	100.0	100.0
LSTM, TBPTT ($k_{\text{trunc}} = 20$)	30.5	35.8	25.7
SAB ($k_{\text{trunc}} = 1$, $k_{\text{top}} = 1$)	57.9	39.9	43.1
SAB ($k_{\text{trunc}} = 5$, $k_{\text{top}} = 5$)	**100.0**	**100.0**	**99.9**

At $T = 300$: $+64.0$ percentage points over full-BPTT LSTM.

Sparse access has enough capacity here; the smallest budget does not.

Paper, Table 1, PDF p. 7. SAB rows include mental updates. This is not the transfer experiment.

Adding improves over TBPTT but trails full BPTT at 400

Reported loss ↓ on summing two masked inputs.

Method	$T = 200$	$T = 400$
LSTM, full BPTT	4.59×10^{-6}	1.554×10^{-7}
LSTM + full attention	5.541×10^{-8}	4.972×10^{-7}
LSTM, TBPTT	3.0×10^{-4}	6.8×10^{-4}
SAB ($k_{\text{trunc}} = 10$, $k_{\text{top}} = 10$)	2.0×10^{-6}	1.001×10^{-5}

TBPTT uses $k_{\text{trunc}} = 50$ at $T = 200$ and $k_{\text{trunc}} = 100$ at $T = 400$.

Metric ambiguity: the table labels continuous-sum loss "CE"; no probability model is specified. Values are retained as reported.

Paper, Table 2 (left), PDF p. 7; task definition in Sec. 7.1, PDF p. 12.

Language modeling largely recovers the truncation gap

Test bits per character (BPC) ↓; values from the main table.

Method	$(k_{\text{trunc}}, k_{\text{top}}, k_{\text{att}})$	PTB	Text8
LSTM, full BPTT	full	**1.36**	**1.42**
LSTM, TBPTT	$(5, -, -)$	1.44	1.56
LSTM, TBPTT	$(20, -, -)$	1.40	—
SAB	$(10, 5, 10)$	1.42	1.47
SAB	$(10, 10, 10)$	1.40	1.45
SAB	$(20, 10, 20)$	1.37	1.44

PTB: SAB nearly matches full BPTT. Text8: full BPTT remains better.

The Text8 prose claims a win over full BPTT; the table contradicts it.

Paper, Table 2 and adjacent prose, PDF p. 7. "Full" uses 100-/180-character training segments.

The Transformer comparison depends on the task

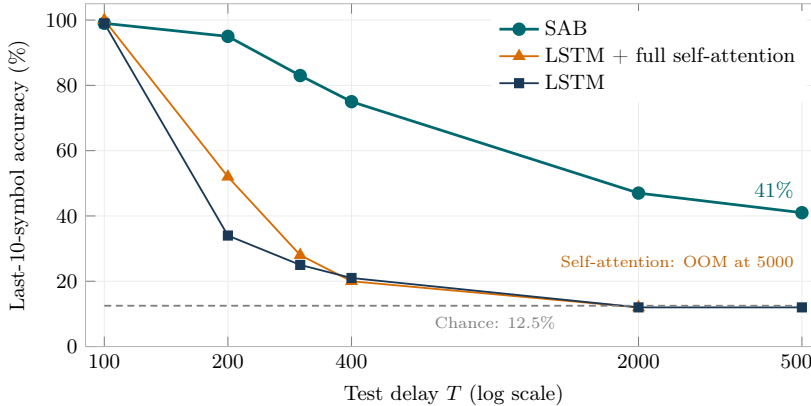
Test accuracy (%) ↑ after sequential pixel input.

Model	pMNIST	CIFAR10
LSTM, full BPTT	90.3	58.3
Best reported SAB configuration	94.2	**64.5**
Transformer	**97.9**	62.2

- pMNIST: SAB uses $(k_{\mathrm{trunc}}, k_{\mathrm{top}}, k_{\mathrm{att}}) = (50, 10, 50)$.
- CIFAR10: SAB uses $(16, 10, 16)$; 1024 input steps, no permutation.
- SAB is $+2.3$ points over Transformer on CIFAR10, but -3.7 on pMNIST.

Paper, Table 4, PDF p. 8. Transformer architecture and budget details are limited in the report.

SAB retains 41% accuracy at 50 times the training delay



Trained at $T = 100$ only. 41% measures retained ability, with substantial errors.

Redrawn from Table 5, PDF p. 8. Logarithmic horizontal axis. OOM is not a zero-accuracy point.

Sparse access and mental updates contribute differently

Copying at $T = 100$: last-10-symbol accuracy (%) ↑.

$(k_{\text{trunc}}, k_{\text{top}})$	With mental updates	Without	Gain (points)
$(1, 1)$	57.9	49.0	+8.9
$(5, 5)$	100.0	98.3	+1.7
$(10, 10)$	100.0	99.6	+0.4

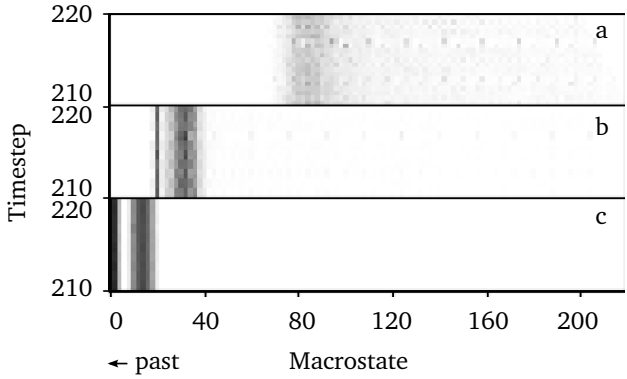
Dense access with truncation is much worse here

$k_{\text{trunc}} = 5$, $k_{\text{top}} = \text{all}$: **40.5%**, versus **100.0%** for SAB $(5, 5)$.

MU helps most at the smallest budget; larger read budgets approach saturation.

Paper, Tables 1 and 3, PDF pp. 7–8. Dense ablation received an extra learning-rate search (Sec. 7.1).

Attention shifts toward the relevant early symbols



- Horizontal: stored history positions; earlier is left.
- Vertical: output-phase time steps.
- Darker: more attention.
- a $\rightarrow$ b $\rightarrow$ c: later snapshots within the first training epoch.

The selected history moves toward the first 10 states needed for copying.

Original attention map from Table 3 (right), PDF p. 8; copying delay $T = 200$. Author vector asset.

Sparse replay leaves growing memory and scoring costs

Let t be sequence length and n hidden-state width.

Paper's complexity	Interpretation
Forward time: $O(t^2 n^2)$	All stored candidates are scored at each step.
Training state space: $O(tn)$	Historical states remain available.
Inference memory: $O(tn)$	Larger than the $O(n)$ state of a plain LSTM.
Backward time: no tight bound	Memories depend on earlier memories; sparse edges form a recursive DAG.

A small number of read edges does not bound total backward work.

The paper establishes learning capability; it reports no wall-clock speedup benchmark.

Paper, Sec. 8, PDF p. 13; unbounded-memory setting and fixed sampling interval.

The main contribution is a design for credit routing

1. **Retrieve and learn through the same temporal shortcut.**
   Attention decides what is read and where useful credit can arrive.

2. **Separate temporal distance from local replay length.**
   This works especially well in the copying experiments and their length transfer.

3. **Treat memory management as a remaining design problem.**
   Content-based writing, bounded storage and cheaper search are natural next steps.

Language-modeling results mostly recover TBPTT losses; image-task wins are mixed.

Paper, Secs. 4–5 and 8, PDF pp. 6–8, 13. Bounded storage and cheaper search are reader priorities.

Paper, publication and implementation

- Ke, Goyal, Bilaniuk, Binas, Mozer, Pal and Bengio.
  Sparse Attentive Backtracking: Temporal Credit Assignment Through Reminding.
  arXiv:1809.03702v1, 2018.

- Official NeurIPS 2018 proceedings entry, volume 31.

- Official implementation: nke001/sparse_attentive_backtracking_release.
  Model layers and copying-task training program are present.
  README lists PyTorch 0.1.12, CUDA 8.0 and cuDNN 6.

All table and page references in this deck use the 13-page arXiv v1 PDF.

Publication and repository verified on 29 October 2026. Figures 1–2 and attention map are author assets.

Backup: the training settings are task-specific

Task	Reported setup
Copying / adding	128 hidden units; Adam; learning rate 10^{-3}. Dense ablation additionally searches 10^{-3} to 10^{-5}.
Character PTB	1000 hidden units; length 100; batch 32; learning rate 0.002; up to 100 epochs with validation early stopping.
Text8	90M / 5M / 5M train / validation / test characters; length 180; batch 64; baselines have 1000 hidden units; SAB up to 30 epochs.
pMNIST	LSTM models: 128 hidden units; learning rate 0.001; about 100 epochs with validation early stopping.

"Full BPTT" in language modeling refers to the stated training segments.

Paper, Secs. 7.1–7.4, PDF p. 12.

Backup: two Text8 inconsistencies affect interpretation

Prose versus Table 2, PDF p. 7

The text says SAB $(10, 5, 10)$ beats full-BPTT LSTM.
The table gives **1.47 versus 1.42 BPC**; lower is better.

Table 2 versus Table 6

SAB $(10, 10, 10)$: **1.45** in the main table, **1.44** in the supplement.
Table 6 also gives dense attention + TBPTT-10 at **1.48**.

- Preserve both SAB values with their own source table.
- The reported evidence favors SAB over truncated baselines.
- It does not show a Text8 win over full BPTT (1.42).

Paper, Table 2 and adjacent prose, PDF p. 7; Table 6, PDF p. 12.

Backup: sparse normalization has derivative edge cases

Reader derivation for $k_{\text{top}} = 1$, away from ranking changes

Only one positive weight remains, so

$$\alpha_{\text{winner}} = \frac{a_{\text{winner}} - \tau}{a_{\text{winner}} - \tau} = 1.$$

The normalized weight is locally constant in the scores.

- Selected memory values still have a direct gradient path.
- Differentiating a score threshold and truncating the replay graph are distinct choices.
- Algorithm 1 leaves empty-memory, too-few-candidate and zero-denominator handling implicit.

Derived from Algorithm 1, PDF p. 5; these observations are not additional author experiments.