SKIPW: RESOURCE ADAPTABLE RNN WITH STRICT UPPER COMPUTATIONAL LIMIT

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce Skip-Window, a method to allow recurrent neural networks (RNNs) to trade off accuracy for computational cost during the analysis of a sequence. Similarly to existing approaches, Skip-Window extends existing RNN cells by adding a mechanism to encourage the model to process fewer inputs. Unlike existing approaches, Skip-Window is able to respect a strict computational budget, making this model more suitable for limited hardware like edge devices. We evaluate this approach on two datasets: a human activity recognition task and adding task. Our results show that Skip-Window is able to exceed the accuracy of existing approaches for a lower computational cost while strictly limiting said cost.

Keywords: Recurrent neural networks, Flexibility, Computational resources.

1 INTRODUCTION

Since Recurrent Neural Networks (RNN) have been introduced Williams et al. (1986), they have become one of the reference methods to process sequences. A typical architecture is the Long-Short-Term-Memory neural network (LSTM) which allowed improvement in natural language processing such as large-vocabulary speech recognition (Sak et al., 2014; Li & Wu, 2014). Used with CNNs they have also reached state of the art in automatic image captioning (Vinyals et al., 2014).

Deep learning models are now brought closer to the user rather than running in a distant cloud, helping to reduce latency, network congestion, and improving data security and privacy. However, smartphones and user devices impose additional constraints such as limited computation or energy. Handling these constraints has become an active research topic (Zhang et al., 2017; 2018; Howard et al., 2019; Wu et al., 2019; Cai et al., 2020). User devices can also host multiple processes running at the same time and starting or stopping abruptly, modifying the constraints affecting the processes. Few works have considered models that can be modified at run time to adapt to an evolving computational limit (Yu et al., 2019; Yu & Huang, 2019; Guerra et al., 2020; Jin et al., 2020). However, none of these focus on sequences and therefore none address the problem of adapting the model in the middle of a sequence.

In this context, this paper introduces Skip-Window (SkipW), a flexible recurrent neural network architecture: its computational cost can be dynamically adapted during a sequence analysis to meet real time constraints changes. The proposed architecture can be combined with any RNN cell and allows to strictly limit the computational resources used to avoid exceeding a given budget. Furthermore, empirical experiments on two data sets (Adding Task and HAR-2D-POSE) demonstrate that Skip-Window is able to exceed the accuracy of existing approaches for a lower computational cost. In addition, measurements on specific processors highlight that SkipW produces computational and energy savings.

2 RELATED WORK

Typically, RNNs maintain a "state", a vector of variables, over time. This state is supposed to accumulate relevant information and is updated recursively. Each input of the sequence is typically a) processed by some deep layers and b) then combined with the previous state through some other deep layers to compute the new state. Hence, the RNN can be seen as a function taking a sequence of inputs $\boldsymbol{x} = (x_1, \ldots, x_T)$ and recursively computing a set of states $\boldsymbol{s} = (s_1, \ldots, s_T)$. Each state

s_t is computed from s_{t-1} and x_t by a cell S of the RNN. As neural networks are increasingly run on limited hardware, recent research has focused on controlling their computational cost.

2.1 Flexible Neural Networks

A few architectures have recently been designed to adapt the computational complexity of a Deep Neural Network (DNN) without reloading the whole model. This can be achieved by removing/adding neurons (Yu et al., 2019; Yu & Huang, 2019) or by modifying the quantization of the weights (Guerra et al., 2020; Jin et al., 2020). An efficient embedding of a mixture of Convolutional Neural Network (CNNs) also allows to add or remove several models at the same time, hence changing the computational cost (Ruiz & Verbeek, 2019).

2.1.1 ThrRnn

For RNNs specifically, ThrRNN (Lambert et al., 2020) aims to control computation time by not processing some inputs. This is controlled by an update gate u_t. The tradeoff between the average accuracy and the average number of updates can be modified during inference by changing a single parameter thr. ThrRNN can wrap any RNN cell S:

$$u_t = f_{binarize}(\tilde{u}_t, thr) = \begin{cases} 0 & if\ \tilde{u}_t < thr \\ 1 & otherwise \end{cases} \tag{1}$$

$$\Delta\tilde{u}_t = \sigma(Ws_t + b) \tag{2}$$

$$\tilde{u}_{t+1} = u_t\Delta\tilde{u}_t + (1 - u_t)(\tilde{u}_t + min(\Delta\tilde{u}_t, 1 - \tilde{u}_t)) \tag{3}$$

$$s_t = u_t S(s_{t-1}, x_t) + (1 - u_t)s_{t-1} \ . \tag{4}$$

When an input is processed, an update gate computes the quantity $\Delta\tilde{u}_t$ that determines how many inputs will be skipped. In practice the $\Delta\tilde{u}_t$ are accumulated in $\tilde{u}_t$ until $\tilde{u}_t \geq thr$.

2.2 Recurrent Neural Network with Low Computational Complexity

Several architectures have been proposed to limit or reduce the computational cost of RNNs, but this cost cannot be adapted at inference. A first class of architectures dynamically reduces computation based on the input. SkipRNN (Campos et al., 2017) predates and is similar to ThrRNN, except that the binarization function does not change. A similar mechanism has been proposed by Zhang et al. (2019). Other architectures directly select the next input to process (Yeung et al., 2016; Yu et al., 2017; Hansen et al., 2019; Song et al., 2018). Early exit has also been investigated by Dennis et al. (2019). Tao et al. (2019) also use $\mathbf{x}_t$ as input to an update gate. So do Seo et al. (2018); Jernite et al. (2017); Li et al. (2020). However, they do not skip any input but perform partial state updates.

A second class of architectures focuses on reducing the overal cost of the RNN. FastRNN is an RNN augmented with a residual connection with two extra scalar parameters and FastGRNN is an improved FastRNN: the residual connection is extended to a gate and RNN matrices are low rank, sparse and quantized (Kusupati et al., 2018). Other architectures reduce the RNN length. Chan et al. (2016) train an encoder to reduce the input length. Yeung et al. (2016); Shan et al. (2018); Chen et al. (2018) propose various mechanisms to summarize subsequences of windows of inputs.

2.3 Recurrent Neural Network with Hierarchical-Dependent Complexity

A class of architectures focuses on hierarchy level concept to reduce the complexity. These methods are mainly used in the context of multi-layer RNNs where each layer is supposed to model a different level in the hierarchy (e.g. for a corpus the levels could be documents, paragraphs, sentences, words, letters). These approaches are based on the fact that a hierarchical separation exists within a sequence of inputs, which might not always be the case.

In (Koutník et al., 2014), the hidden state is partitioned into different modules, each module has its own clock period, meaning that they will be updated at different times. Skipping updates of part of the hidden state decreases the computational cost. In (Koutník et al., 2014), the update periods are chosen arbitrarily, for example using an exponential series.

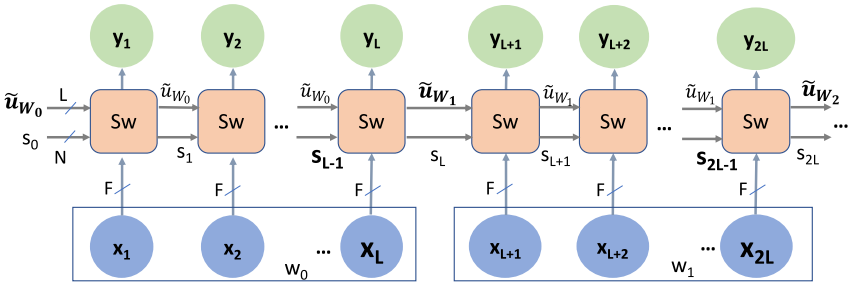

Figure 1: Skip-window computes update gates at the beginning of each window of inputs.

For stacked-RNNs, Chung et al. (2016); Chang et al. (2017) conditionally update each layer based on a feature level criterion, or by dilating a skip connection. Layers close to the inputs would model lower feature levels and be updated more frequently. Further layers would model higher level features. In (Chung et al., 2016), a layer modeling sentences would be updated only when a word is processed entirely (in a model fed character by character), from the layer modeling words. Before the end of a word is reached, the state of the former layer is copied across input steps.

2.4 Relationship to our work

ThrRNN is the closest model to SkipW. Both are flexible RNNs and skip some inputs. However, ThrRNN optimizes computational cost on average over sequences. This induces two variabilities: a) inter-sequence variability: the model will not use the same number of updates for every sequence; and b) intra-sequence variability: the number of updates will not be uniform across time steps, updates may be concentrated in a certain part of the sequence. These two variabilities can cause the model to exceed its computational budget and, therefore, to either shut down or delay the output. SkipW does not have this problem as it strictly enforces a computational constraint over each window of inputs.

Other strategies for flexible models are not straightforward to apply to RNN. They require specialized training algorithms. They have never been applied to models processing inputs of an RNN or to make an RNN flexible and it is not clear how they would need to be modified. Furthermore, these models adapt between independent inputs whereas, for sequences, adaptation is necessary between time steps. RNN architectures with low complexity are orthogonal to our approach. They do not offer flexibility. They could be combined with and benefit from our approach. However, SkipRNN (which we are based on) and related methods have one big advantage over others: by skipping inputs, they also skip any modification of an input, such as processing by an expensive CNN for images. As SkipW makes decision over a window of inputs, it has some superficial similarity to methods summarizing windows or hierarchical RNNs. However, SkipW a) does not summarize windows and b) does not even look at these inputs before deciding what to skip.

3 Skip-Windows architecture

Skip-Windows (SkipW) is a wrapper for a RNN cell S. It uses a conditional computation mechanism to skip some updates. Rather than at each input x_t, update gates are computed at the beginning of windows of inputs, that is, every L time steps (Figure 1). In other words, before any new L-size window of inputs, a L-size vector $\tilde{u}_W$ is computed. $\tilde{u}_W[i]$ can be seen as the importance of input i in the window.

Then, the architecture includes a $select_K$ mechanism. This function takes as input the vector $\tilde{u}_W$ and outputs the vector $\tilde{u}_W^K$, setting $L - K$ elements to a value that ensures the associated inputs are not processed (0 in Figure 2). Therefore, it ensures that at most K out of every L inputs will be processed. In other words, it forces the RNN cell to skip $(L-K)$ out of every L inputs. This ensures a strict upper bound on the computational cost of the model for a sequence and for each window, therefore alleviating the inter-sequence variability and intra-sequence variability issue.

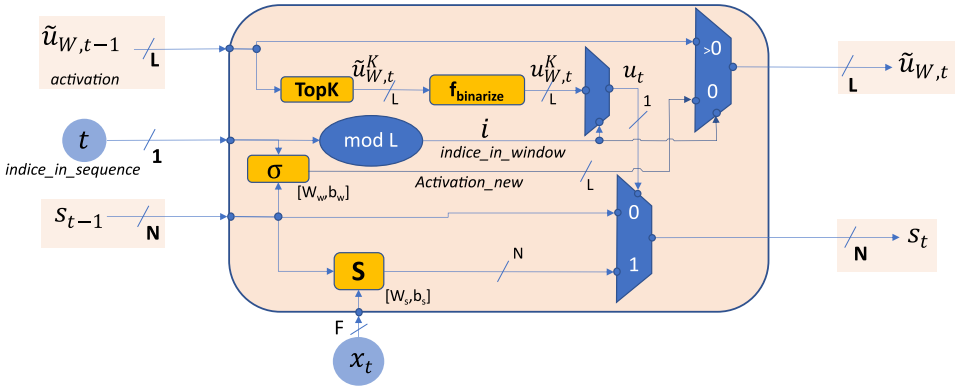

Figure 2: TopK-based Skip-Window cell implementation example.

Similarly to other works, the binary state update L-size vector, u_W, is then obtained by binarizing the remaining values as in equation 1. For example, by setting all values below a threshold to a value that ensures the associated inputs are not processed (0 in Figure 2).

An example of the Skip-Window cell implementation is represented in Figure 2. In this case, $select_K$ is implemented as a top_K function. This enforces the strict constraint on the number of updates. The top_K operation keeps unchanged the K highest value in $\tilde{u}_{W,t}$, and sets the $(L-K)$ others to 0.

The corresponding architecture can be characterized as follows:

$$s_t = u_t \cdot S(s_{t-1}, x_t) + (1 - u_t) \cdot s_{t-1} \tag{5}$$

$$\tilde{u}_{W,t+1} = \gamma \cdot \sigma(W_w(s_{t-1}, t) + b_w) + (1 - \gamma) \cdot \tilde{u}_{W,t} \tag{6}$$

$$\gamma = \begin{cases} 1 & if\ i == 0 \\ 0 & otherwise \end{cases} \tag{7}$$

$$i = t\ mod\ L \tag{8}$$

$$\tilde{u}^K_{W,t} = top_K(\tilde{u}_{W,t}) \tag{9}$$

$$u_t = f_{binarize}(\tilde{u}^K_{W,t}[i], thr) = \begin{cases} 0 & if\ \tilde{u}^K_{W,t}[i] < thr \\ 1 & otherwise \end{cases} \tag{10}$$

where W_w is a weight matrix of size $(N+1) \times L$, N is the number of hidden units as defined by the RNN cell S, b_W is a L-vector bias, σ is the sigmoid function and mod is the modulo operation. Instead of a top_K function for $select_K$, it is also possible to use a stochastic sampling mechanism that randomly selects (without replacement) K out of L elements of $\tilde{u}_W$ where the probability of selecting each element of index i is proportional to $\tilde{u}_W[i]$.

Including the time step t in equation 6 is also optional and can be replaced by a value ensuring the state is not static if no update is made in a window. For example, the number of inputs since the last update or the number of windows already processed.

Training the model The model is trained to minimize a two-part loss, similarly to Campos et al. (2017). The first term measures the accuracy of the task, and the second one penalizes inputs used:

$$L_{budget} = \lambda \sum_{t=1}^{T} u_t \ , \tag{11}$$

where λ is the cost associated to the use of a single input.

Error gradients The model is differentiable except for the $f_{binarize}$ and top_K function. To train the model using standard backpropagation the straight-through estimator is used as done in Campos et al. (2017) for $f_{binarize}$.

Other alternatives might involve reinforcement learning such as REINFORCE (Williams, 1992) or, in the case of the top_K function, the usage of a differentiable top_K as proposed in Xie et al. (2020).

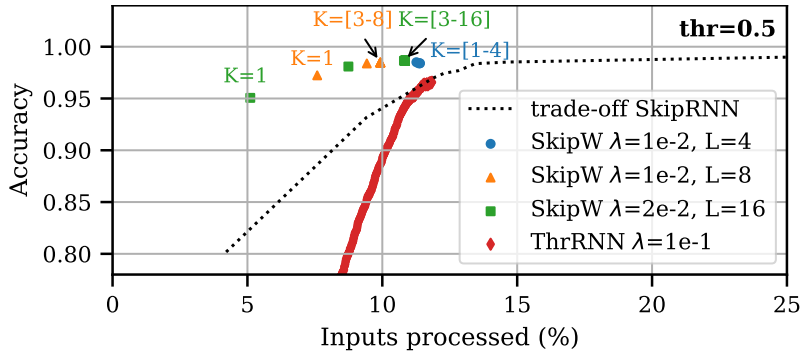

Figure 3: Impact of the K parameter on accuracy/updates tradeoff for HAR. Comparison between ThrRNN (thr=0.5 to 1.0), SkipW(L=4,8,16 , thr=0.5, K=1 to L) and SkipRNN (λ=1e-3 to 3e-1).

Early experiments using a differentiable top_K (Xie et al., 2020) have shown worse results than the straight-through estimator. This suggests that constraining computation may be an interesting approximation for a top_K operation.

Adapting computational cost at inference During inference, adapting the tradeoff between model performance and computational cost can be done using two factors : the K in equation 9 and the thr in equation 10. These two parameters can be modified together or one at a time. Increasing/lowering thr parameter in $[0, 1]$ encourages the model to process fewer/more inputs. Changing K in $\{0 \dots L\}$ forces the model to process at most K/L of the window.

Choice of the window size hyper-parameter By the nature of the model, the task can influence the choice of L. It can be hand tuned or computed using typical hyper-parameter search methods such as grid search. Choosing small L allows the model to make update decisions for the near future only but offers less choice in operating points. Similarly a bigger L requires the model to predict its update decisions for a bigger time span but offers more flexibility. At the extreme when $L = 1$, each window consists of a single input.

4 Experiments

We empirically show that SkipW a) learns accuracy / computational tradeoffs on par with or, surprisingly, sometimes better than our baselines SkipRNN and ThrRNN; b) upper bounds computational cost; c) enables changing the tradeoff in the middle of a sequence. To evaluate the computational benefit induced by skipping part of the input sequence we use the number of inputs processed by the model, as this measure is independent from the hardware and the framework used to implement the models. In addition, we also show how a reduction of the number of inputs processed translate into computational savings on tiny devices.

We evaluate our approach on two data sets. **Human Activity Recognition (HAR)** (Ofli et al., 2013) consists of 2D-position sequences of 18 joints across a 32-frames videos recorded at 22Hz ($\approx$1.5 seconds). Each sequence is labeled by one of the 6 actions performed by 12 subjects filmed from 4 angles. **Adding Task** Hochreiter & Schmidhuber (1997), consists of tuples-sequences (value, marker) $\in \mathbb{R} \times \{0, 1\}$, where the expected output is $\sum_i marker \times value$. For HAR, two stacked RNNs are used, but only one for Adding Task. More details can be found in Appendix A.

4.1 Accuracy and computational tradeoffs

In Figures 3 and 4, the baselines SkipRNN and ThrRNN are compared to SkipW with various values for L and K. Small L values may not offer good tradeoffs when varying K. See for example $L = 4$ in Figure 3. That's not surprising: often the number of inputs processed can be significantly reduced without affecting accuracy. For example, SkipRNN on HAR can process as few as 15% inputs without affecting accuracy. Using $L = 4$ and $K = 1$ limits complexity at 20% only. So, for such a

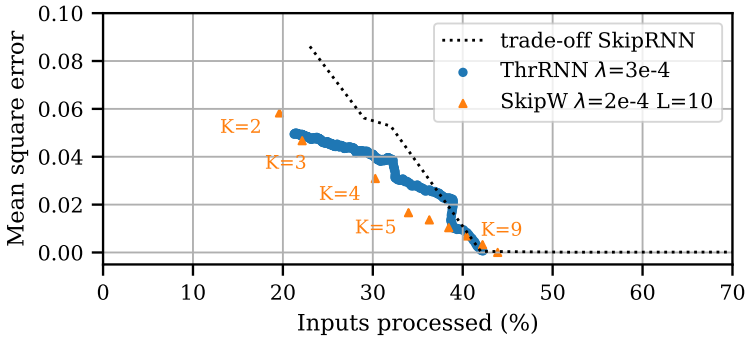

Figure 4: Impact of the parameter K on mean square error and inputs processed for Adding Task. Comparison between ThrRNN ($\lambda = 3e-4$, $thr = 0.5$ to 1.0) and SkipW ($\lambda = 2e-4$, L=10, thr=0.5, K=1 to L). The dotted line corresponds to the performance of individual SkipRNN models trained with different values of λ (from $\lambda = 1e-5$ to $\lambda = 5e-4$).

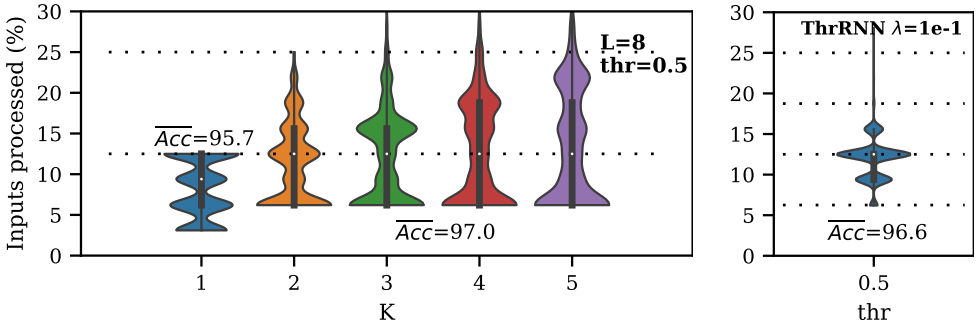

Figure 5: Left: Impact of the K parameter on the computational cost upper limit (SkipW: λ=6e-2, L=8, thr=0.5). Right: ThrRNN with a similar average inputs processed is provided for comparison.

small L, varying K has almost no impact on average. Larger window sizes, on the other hand, offer more tradeoffs when changing K. For example, on HAR, for $L \in \{8, 16\}$, K can take more values, and for small K computational gains are traded off for accuracy. For example, for $L = 16$, reducing K from 4 to 1 more than halves the number of inputs processed for a drop of 3.6% in accuracy. A similar behavior can be observed on Adding Task.

SkipW enforces a K/L upper bound on the number of inputs processed in each window. Even when the average number of updates is not impacted much by a change in K, individual sequences can be impacted. Figure 5 illustrates this impact with a violin plot of the number of inputs processed per sequence for various K and $L = 8$ on HAR. Decreasing K below 3 impacts the maximum observed number of updates over the whole sequence. However, even changes above $K = 3$ have an impact. Even though the observed maximum number of updates does not change, the violin changes as some sequences have fewer inputs processed. This is due to the fact that the computational constraint is enforced over each window. So, for $K \leq 5$, some sequences are prevented from exceeding the computational budget in some windows of inputs.

It is perhaps surprising that SkipW achieves both a higher accuracy and lower computational cost than SkipRNN and ThrRNN, as the former decides on the next K inputs to process at once whereas the baselines decide on the next input only and so decide more often and with more information. We have observed that the baselines tend to process one input regularly, even if there is no information in the signal. This increases the computational cost without benefiting accuracy. We believe this is due to the increment mechanism (Equation 3). To skip C inputs, $\Delta\tilde{u}_t$ (Equation 2) must be smaller than $0.5/C$. We conjecture that at some point, for large C, the model is unable to differentiate between these small values and so no more than C skips are performed.

SkipW is able to learn where relevant inputs are and tends not to use irrelevant inputs. This is best illustrated on AddingTask. Let m_1 be the index of the first marker, and m_2 the index of the second

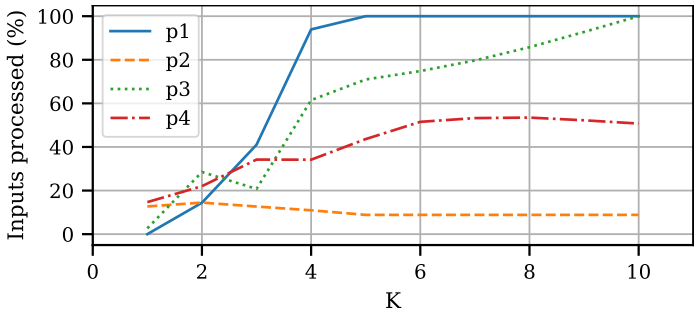

Figure 6: Proportion of inputs processed for different parts p_i of the sequence defined by equations 12 and 13 while changing K with $thr = 0.5$, for a model trained with $\lambda = 2e - 4$ and $L = 10$. The values at $K = L$ indicate the training configuration.

one. The inputs can be divided into four parts $p_i, i \in \{1, 2, 3, 4\}$:

$$p1 = (x_i \mid 0 \leq i \leq m_1) \qquad p2 = (x_i \mid m_1 < i < 25) \tag{12}$$

$$p3 = (x_i \mid 25 \leq i \leq m_2) \qquad p4 = (x_i \mid m_2 < i < 50) \tag{13}$$

Ideally the model should skip all inputs in p_2 and p_4 where there is no marker, while using all the inputs in p_1 and in p_3 where a marker can appear.

Figure 6 illustrates that SkipW skips inputs in different parts of the sequence for various K. For $K = L$ (training point), almost no inputs are skipped in p_1 and p_3, where the markers are, but some are skipped in p_2 and p_4, where there is no marker. Not all inputs are skipped in p_2 and p_4. That's not surprising: decisions to skip are taken at the start of a window, so even if a marker is present in the first input of the window, SkipW will not update its decision to process other inputs in the window. Values of K smaller than but close to L force the model to skip some inputs. Skips are still mainly made in part p_2 and p_4 for these K values but start increasing in p_3 as the model can no longer process all inputs. Finally, when $K < 5$, SkipW can no longer process the first 5 inputs and skips in p_1 start happening. Note that both the binarization function and the inclusion of t in the inputs of the update gate are necessary for this behavior. Please see the ablation study in Appendix C.4.1 for more information.

Modifying thr impacts the accuracy / processing tradeoff as well. Increasing thr gradually reduces both accuracy and average computational cost. This is similar to ThrRNN. Changing thr does not affect the upper bound on computational cost but can provide more accuracy / processing tradeoff points. Some examples are provided in Appendix B.

4.2 Modifying the tradeoff during inference

Modifying (thr, K) is also possible during the analysis of a sequence. Results of a change between two different pairs of (thr, K) values are shown in Figure 7. The average accuracy and computational cost of the analysis of the sequences in the test set are reported as a function of the position of the change. The position at the bottom left (resp. top right) of the graph corresponds to the operating point of tradeoff$_2$ (resp. trade˙off$_1$) when it processes the whole sequence.

The accuracy typically remains between the accuracy of the individual models and increases almost monotonically from the values of tradeoff$_2$ to the values of tradeoff$_1$. This suggests that SkipW can switch between operating points during the analysis of a sequence. This behavior was observed on all data sets, for various training runs and (thr, K) configurations. More details are provided in the supplementary material (C and D).

4.3 Real time experiments

We evaluate the performance of SkipW on small hardware to showcase its performance in the setting it is designed for. We implement the full service, from images to activity recognition, on a Nvidia Jetson Nano platform. In the HAR-2D-POSE data set, OpenPose is used to compute poses on images, but the embedded GPU is not powerful enough to run this OpenPose. Instead, we choose to

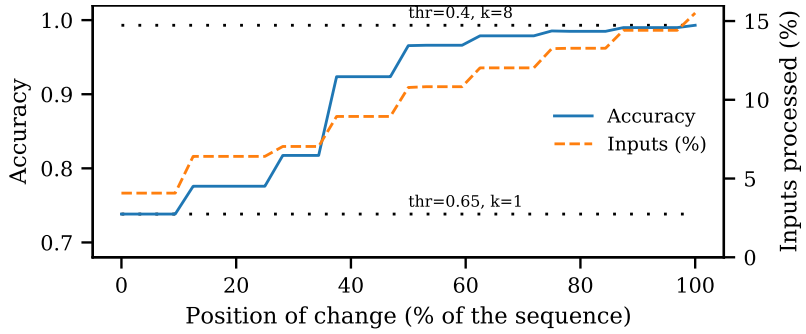

Figure 7: Modifying (thr, K) during the sequence on HAR for SkipW ($\lambda = 1e-2$, $L = 8$): from tradeoff$_1$($thr = 0.4$, $K = 8$, acc/updates tradeoff = 99.3%/15.5%) to tradeoff$_2$($thr = 0.65$, $K = 1$, acc/updates tradeoff=73.8%/4%).

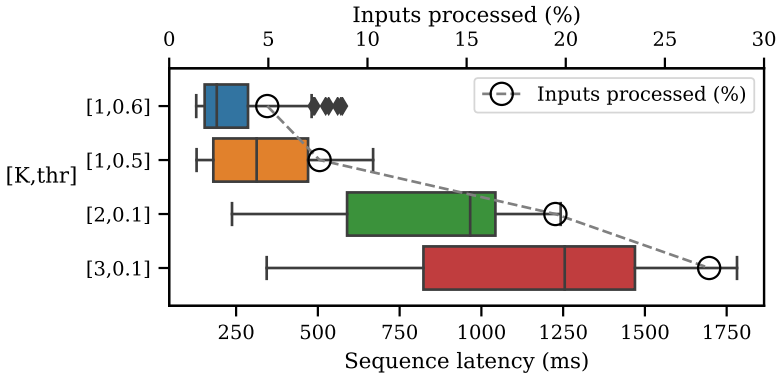

Figure 8: Boxplots of inference latency time for a HAR Service based on SkipW ($\lambda = 1e-2$, $L = 8$) coupled with a Posenet model running on Jetson Nano.

use PoseNet (MobileNetV1 architecture with a 0.75 multiplier) model which is much more efficient to run on such a small platform. In our implementation, PoseNet model running on the embedded GPU takes roughly 120 ms to compute a pose on a 640x480 pixel image, while the SkipW model takes around 30 ms to perform all state updates for a sequence.

Figure 8 presents the individual inference time of 1000 HAR image sequences processed by SkipW coupled with the PoseNet detection model on a Jetson Nano platform. As expected, when K decreases, the run time decreases sharply. This gain is largely due to the lower number of pose detections that need to be computed. By drastically limiting the number of images used as input, SkipW saves a large amount of time, computation and energy. Results on Jetson TX2 and Raspberry Pi4 lead to similar conclusions (Appendix E).

5 Conclusion

We presented Skip-Window, a novel RNN cell architecture that dynamically controls the number of inputs processed during the analysis of a sequence. SkipW allows to adapt the computational cost during and between sequence analysis without reloading any weight. Unlike existing flexible RNNs, SkipW can enforce a strict upper bound on the number of updates performed over a time interval. Therefore, SkipW can operate without exceeding a given computational budget. Furthermore, our results show that Skip-Window is able to exceed the accuracy of existing flexible RNNs for a lower computational cost.

The strict limit over computational resources across a time interval makes SkipW well suited for shared hardware with real time constraints, as we have shown that SkipW indeed reduces computation on such hardware.

REFERENCES

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HylxE1HKwS.

Victor Campos, Brendan Jou, Xavier Giró-i-Nieto, Jordi Torres, and Shih-Fu Chang. Skip RNN: learning to skip state updates in recurrent neural networks. *CoRR*, abs/1708.06834, 2017. URL http://arxiv.org/abs/1708.06834.

William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4960–4964. IEEE, 2016.

Shiyu Chang, Yang Zhang, Wei Han, Mo Yu, Xiaoxiao Guo, Wei Tan, Xiaodong Cui, Michael J. Witbrock, Mark Hasegawa-Johnson, and Thomas S. Huang. Dilated recurrent neural networks. *CoRR*, abs/1710.02224, 2017. URL http://arxiv.org/abs/1710.02224.

Mengzhe Chen, Shiliang Zhang, Ming Lei, Yong Liu, Haitao Yao, and Jie Gao. Compact feed-forward sequential memory networks for small-footprint keyword spotting. In *Interspeech*, pp. 2663–2667, 2018.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. *CoRR*, abs/1609.01704, 2016. URL http://arxiv.org/abs/1609.01704.

Don Dennis, Durmus Alp Emre Acar, Vikram Mandikal, Vinu Sankar Sadasivan, Venkatesh Saligrama, Harsha Vardhan Simhadri, and Prateek Jain. Shallow rnn: accurate time-series classification on resource constrained devices. In *Advances in Neural Information Processing Systems*, pp. 12896–12906, 2019.

Stuart Eiffert. RNN for human activity recognition - 2d pose input, 2018. URL https://github.com/stuarteiffert/RNN-for-Human-Activity-Recognition-using-2D-Pose-Input.

Luis Guerra, Bohan Zhuang, Ian Reid, and Tom Drummond. Switchable precision neural networks. *arXiv preprint arXiv:2002.02815*, 2020.

Christian Hansen, Casper Hansen, Stephen Alstrup, Jakob Grue Simonsen, and Christina Lioma. Neural speed reading with structural-jump-LSTM. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=B1xf9jAqFQ.

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

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1314–1324, 2019.

Yacine Jernite, Edouard Grave, Armand Joulin, and Tomas Mikolov. Variable computation in recurrent neural networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Sy-dQG-Rb.

Qing Jin, Linjie Yang, and Zhenyu Liao. Adabits: Neural network quantization with adaptive bit-widths. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2146–2156, 2020.

Jan Koutník, Klaus Greff, Faustino J. Gomez, and Jürgen Schmidhuber. A clockwork RNN. *CoRR*, abs/1402.3511, 2014. URL http://arxiv.org/abs/1402.3511.

Aditya Kusupati, Manish Singh, Kush Bhatia, Ashish Kumar, Prateek Jain, and Manik Varma. Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network. In *Advances in Neural Information Processing Systems*, pp. 9017–9028, 2018.

Anne Lambert, Françoise Le Bolzer, and François Schnitzler. Flexible recurrent neural networks. In *Machine Learning and Knowledge Discovery in Databases*, 2020.

Xiangang Li and Xihong Wu. Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition. *CoRR*, abs/1410.4281, 2014. URL http://arxiv.org/abs/1410.4281.

Yikang Li, Tianshu Yu, and Baoxin Li. Recognizing video events with varying rhythms. *arXiv preprint arXiv:2001.05060*, 2020.

Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. Phased lstm: Accelerating recurrent network training for long or event-based sequences. In *Advances in Neural Information Processing Systems*, pp. 3882–3890, 2016.

F. Ofli, R. Chaudhry, G. Kurillo, R. Vidal, and R. Bajcsy. Berkeley mhad: A comprehensive multimodal human action database. In *2013 IEEE Workshop on Applications of Computer Vision (WACV)*, pp. 53–60, 2013.

Adria Ruiz and Jakob J. Verbeek. Adaptative inference cost with convolutional neural mixture models. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 1872–1881, 2019.

Hasim Sak, Andrew W. Senior, and Françoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *CoRR*, abs/1402.1128, 2014. URL http://arxiv.org/abs/1402.1128.

Minjoon Seo, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. Neural speed reading via skim-RNN. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=Sy-dQG-Rb.

Changhao Shan, Junbo Zhang, Yujun Wang, and Lei Xie. Attention-based end-to-end models for small-footprint keyword spotting. In *Proc. Interspeech 2018*, pp. 2037–2041, 2018. doi: 10.21437/Interspeech.2018-1777. URL http://dx.doi.org/10.21437/Interspeech.2018-1777.

Inchul Song, Junyoung Chung, Taesup Kim, and Yoshua Bengio. Dynamic frame skipping for fast speech recognition in recurrent neural network based acoustic models. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4984–4988. IEEE, 2018.

Jin Tao, Urmish Thakker, Ganesh Dasika, and Jesse Beu. Skipping rnn state updates without retraining the original model. In *Proceedings of the 1st Workshop on Machine Learning on Edge in Sensor Systems*, pp. 31–36, 2019.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. *CoRR*, abs/1411.4555, 2014. URL http://arxiv.org/abs/1411.4555.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, May 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

Ronald J. Williams, Geoffrey E. Hinton, and David E Rumelhart. Learning representations by back-propagating errors. In *Nature 1986*, pp. 533—-536, 1986. doi: 10.1038/323533a0. URL https://ui.adsabs.harvard.edu/abs/1986Natur.323..533R/abstract.

Carole-Jean Wu, David Brooks, Kevin Chen, Douglas Chen, Sy Choudhury, Marat Dukhan, Kim Hazelwood, Eldad Isaac, Yangqing Jia, Bill Jia, et al. Machine learning at facebook: Understanding inference at the edge. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 331–344. IEEE, 2019.

Yujia Xie, Hanjun Dai, Minshuo Chen, Bo Dai, Tuo Zhao, Hongyuan Zha, Wei Wei, and Tomas Pfister. Differentiable top-k operator with optimal transport, 2020.

Serena Yeung, Olga Russakovsky, Greg Mori, and Li Fei-Fei. End-to-end learning of action detection from frame glimpses in videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2678–2687, 2016.

Adams Wei Yu, Hongrae Lee, and Quoc Le. Learning to skim text. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1880–1890, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1172. URL https://www.aclweb.org/anthology/P17-1172.

Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1803–1811, 2019.

Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=H1gMCsAqY7.

Shucong Zhang, Erfan Loweimi, Yumo Xu, Peter Bell, and Steve Renals. Trainable dynamic subsampling for end-to-end speech recognition. *Proc. Interspeech 2019*, pp. 1413–1417, 2019.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6848–6856, 2018.

Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. Hello edge: Keyword spotting on microcontrollers. *CoRR*, abs/1711.07128, 2017. URL http://arxiv.org/abs/1711.07128.

A EXPERIMENTAL SETTINGS DETAILS

We evaluate our approaches on two data sets:

- **Human Activity Recognition (HAR):** HAR is an action recognition dataset consisting of 2D pose estimations extracted from videos (Eiffert, 2018). The 2D pose estimation was performed on a subset of the Berkeley Multimodal Human Action Database (MHAD) dataset (Ofli et al., 2013). Six actions (Jumping in place, Jumping jacks, Punching (boxing), Waving two hands, Waving one hand, Clapping hands) are considered for a total of 1438 videos recorded at 22Hz. Each action must then be identified based on a sequence of 32 frames ($\approx$ 1.5 sec). Each video gives several sequences by applying an overlapping ratio. A pose is extracted from each frame and estimated by 18 body joints expressed in a 2D coordinate system. Following the parameter setting defined in (Eiffert, 2018), the dataset is split into 2 independent partitions : 22625 sequences for training and 5751 for validation.

  The model architecture consists of a two-stacked RNN of 60 GRU cells each followed by a fully connected layer with a RELU activation function. The model is trained with batches of 512 sequences using a decaying learning rate for 600 epochs. The best performance was obtained when the model was initialized with a model trained with a smaller λ.

  This architecture was selected by performing an optimization over the hyperparameters. The following parameters were included in the search:

  - Batch size: 4096 and 512
  - $\lambda \in \{1e-4, 1e-3, 1e-2\}$
  - Cell type: LSTM or GRU
  - Number of cells $\in \{30, 40, 50, 60\}$ per layer (identical number of cells in each layer)
  - Window size $L \in \{4, 8, 16\}$ (SkipW only)

  The search was performed both for ThrRNN (baseline) and SkipW (our approach). Several combinations of parameters resulted in models that are Pareto optimal with respect to accuracy and computational cost. To evaluate SkipW and compare it to ThrRNN, we selected one parameter configuration that led to models that were Pareto optimal (or close to Pareto optimal) for ThRNN and for several values of L for SkipW.

- **Adding Task:** This is a synthetic dataset consisting of sequences of (value, marker) tuples, where the expected output is $\sum_i marker \times value$. This dataset is commonly used to evaluate the performance of RNN models [(Hochreiter & Schmidhuber, 1997), (Neil et al., 2016), (Campos et al., 2017)]. The $value$ elements are uniformly sampled in the range (-0.5, 0.5) and the marker elements are equal to 1 or 0. We reproduced the experimental setup defined in (Neil et al., 2016) with a sequence of length 50. Two values are marked for addition with the first marker randomly placed in the first 10% of the sequence and the second one in the last half of the sequence.

  The model architecture consists of a single layer of 110 LSTM cells and a fully connected layer regressing the scalar output. We train the model with a batch size of 256 and a learning rate of $1e-4$. As with HAR dataset, the best performance was obtained when the model was initialized with a model trained with a smaller λ.

B IMPACT OF THR

Modifying thr impacts the accuracy / processing tradeoff as well. However, changing thr does not affect the upper bound on computational cost. Increasing thr gradually reduces both accuracy and average computational cost. This is similar to ThrRNN. Figure 9 illustrates the impact of changing thr for various L on HAR. While this is less interesting in the context of a strict limit on computation, it increases the variety of the possible tradeoff's points. That being said, the thresholding mechanism allows the model to skip uninteresting inputs and is therefore necessary. For example, if this mechanism is removed on adding task, SkipW processes inputs in parts of the sequence where there is no marker. On the other tasks, removing it similarly increases computational cost without improving accuracy. This is better highlighted in Appendix C.4.1.

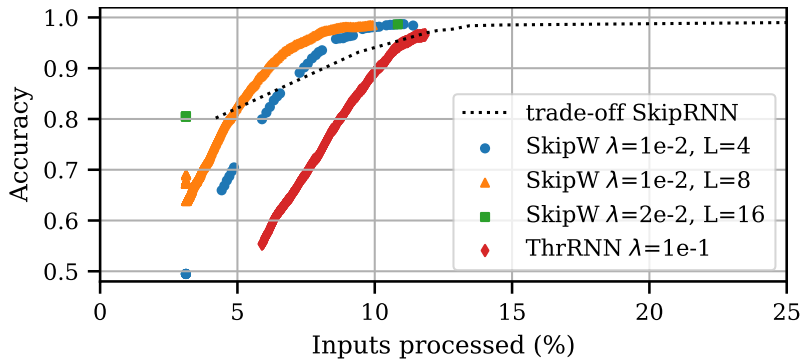

Figure 9: Impact of the thr parameter on accuracy/updates tradeoff for HAR. Comparison between ThrRNN ($thr = 0.5$ to 1.0), SkipW ($thr = 0.5$ to 1.0, $K = L$) and SkipRNN ($\lambda = 1e-3$ to $3e-1$).

C Detailed results for Adding task

The Adding task dataset penalizes the model when skipping markers. Therefore, a successful model will learn to process as many inputs as possible where the makers can be placed and to ignore the parts of the sequence where the markers are never found.

C.1 Accuracy and computational tradeoffs

As we see in Figure 4, both SkipW and ThrRNN offer operating points that are better than the ones obtained with individual SkipRNN models (dotted line in the figure). SkipW offers results with a lower number of updates and lower error than ThrRNN while adapting thr especially for K between 5 and 10.

C.2 Modifying tradeoff during inference

We test SkipW adaptability during the analysis of a sequence; switching between a model with a low MSE, high update rate and another model with a higher MSE and lower update rate.

As illustrated by Figure 10, the evolution of the MSE is not as smooth as in the HAR experimentation due to the nature of the task - skipping a marker greatly penalizes the MSE. We can note that changes in MSE and number of inputs processed happen when the switch is performed in the last 40% of the sequence. This is explained by the fact that the distribution of the inputs processed is identical for $K = 5$ and $K = 10$ at the beginning of the sequence. Both models a) process the first 5 inputs, to find the first marker, b) skip other inputs up to 50% of the sequence, as they never contain a marker and c) process the last 5 inputs of the window containing the middle of the sequence, as the second marker may be there. This is illustrated in Figure 12 for $K = 10$ and Figure 14 for $K = 5$.

As the loss in MSE stays reasonable proportionally to the number of updates skipped, SkipW can be used to reduce the number of updates during a sequence to fulfill computational restriction.

C.3 Computational Tradeoffs

Figure 11 illustrates the ability of SkipW to strictly limit the number of inputs processed. The dotted lines corresponds to the K/L theoretical limit imposed. It is worth noting that the model uses a maximum of 60% of inputs for $K = L$. This can be explained by the nature of the task where 40% of the sequence does not contain information (no marker) and is also illustrated by the distribution of inputs processed in Figure 12 (SkipW, top row).

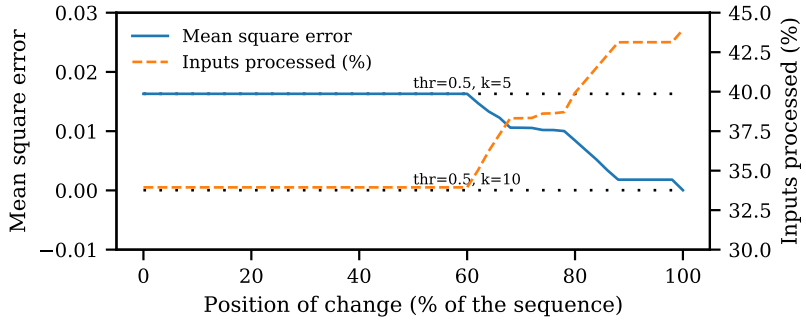

Figure 10: Modifying K during the analysis of a sequence on Adding Task for SkipW ($\lambda = 2e-4$, $L = 10$): from tradeoff$_1$($thr = 0.5$, $K = 10$, MSE/updates = 3.56e-5/44.4%) to tradeoff$_2$($thr = 0.5$, $K = 5$, MSE/updates=1.66e-2/34.2%).

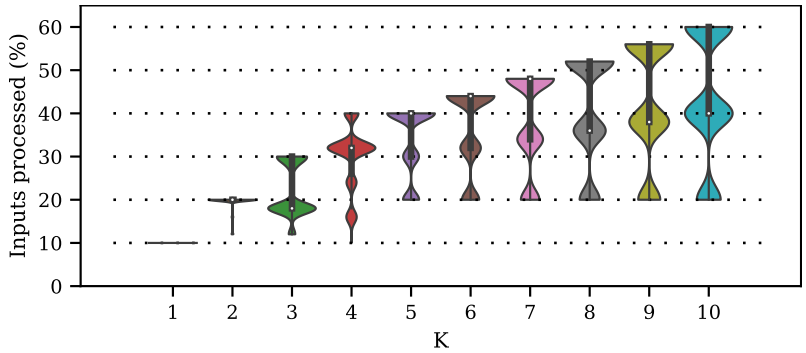

Figure 11: Impact of the K parameter on the computational cost upper limit (SkipW: $\lambda = 2e-4$, $L = 10$, $thr = 0.5$).

C.4 Impact of architecture elements on the distribution of inputs processed in a sequence

C.4.1 Ablation of index in sequence (t)

As discussed in Section 3, including the index (t) in the input of the update gate (Equation 6) is necessary to allow the model to skip all inputs in a window. We show here the impact of removing it by modifying equation 6: $\tilde{u}_{W,t+1} = \gamma \cdot \sigma(W_w(s_{t-1}) + b_w) + (1-\gamma) \cdot \tilde{u}_{W,t}$.

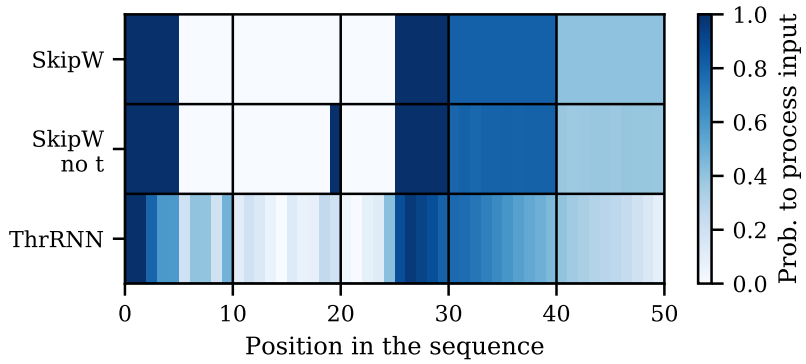

Figure 12: Distribution of the inputs processed for Adding Task using different models with $thr = 0.5$. From top to bottom: SkipW $K = L = 10$ and $\lambda = 2e-4$, SkipW with no index in sequence $K = L = 10$ and $\lambda = 3e-4$, ThrRNN $\lambda = 3e-4$.

Table 1: Inputs processed in different part of sequence for Adding Task

Architecture	first half	second half
SkipW ($L = 10$, $\lambda = 2e - 4$)	20%	69%
ThrRNN ($\lambda = 3e - 4$)	31%	54%

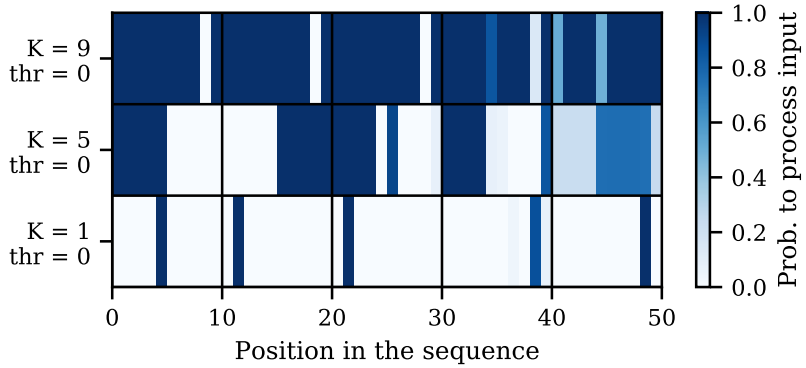

Figure 13: Distribution of the inputs processed for Adding Task using a SkipW model trained with $thr = 0.5$, $L = 10$ and $\lambda = 2e - 4$; inference performed with $thr = 0$ and different values of K.

Figure 12 shows that, without t, the model is forced to process an additional input in the second window even though this part of the sequence contains no marker. Processing this input enables the model to change its hidden state and therefore to obtain a different activation vector $\tilde{u}_{W,t+1}$ in the third window and to continue processing the second part of the sequence where the second marker is found. Without this update, the model would stop processing the inputs at the end of the first window since the state and therefore the activation vector would not change. Including t in the input of the activation gate ensures that a different activation vector may be computed even without processing an input in a window.

Figure 12 also compares the SkipW models with ThrRNN. Notice that ThrRNN is unable to stop processing inputs for a long period, as shown by a non-zero probability of using inputs between position 5 and 25. In addition, ThrRNN sometimes skip the first potential location of the second marker. We believe this explains why SkipW can achieve both a higher accuracy and a lower computational cost than SkipRNN and ThrRNN. On the other hand, contrary to SkipW models who determine the inputs to process at the beginning of each window, the probability of processing an input gradually lowers after position 25 for ThrRNN making it more efficient for the second part of the sequence as shown in Table 1. The same behavior can be observed for SkipW, but only from one window to another. It is also worth noting that the probability that SkipW processes inputs in a window is very close to the probability that ThrRNN processes the first input in the window.

C.4.2 Ablation of threshold (thr = 0)

A second experiment was done to show the importance of the threshold parameter. When the threshold is set to 0 the inputs are selected sorely on the basis of the parameter K. Therefore exactly K inputs are selected in each window using equation 9 and equation 10 becomes $u_t = f_{binarize}(\tilde{u}^K_{W,t}[i]) = 1$ if $\tilde{u}^K_{W,t}[i] > 0$, 0 otherwise.

Figure 13 shows the distribution of the inputs in each window for different values of K when $thr = 0$ and Figure 14 when $thr = 0.5$. The importance of thr is clearly highlighted when comparing the two figures. For example, for $K = 5$ and $thr = 0$ (Figure 13), there is no difference in the first window and the model finds the first marker. However, as the model is forced to use 5 inputs per window, it processes inputs in the second window and this impacts the choice made in the third window as the model processes inputs at the beginning of this window instead of the last 5 which could contain the second marker. This behavior is to be compared with Figure 14 ($thr = 0.5$) where the inputs processed are effectively located in the second half of the third window. It should be noted that for $K = 1$, thr has no discernible effect. The model selects only one input in the first

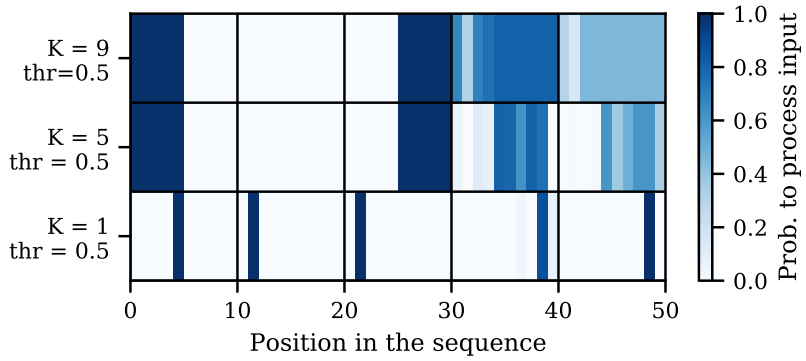

Figure 14: Distribution of the inputs processed for Adding Task using a SkipW model trained with $L = 10$ and $\lambda = 2e - 4$ with $thr = 0.5$ and different values of K.

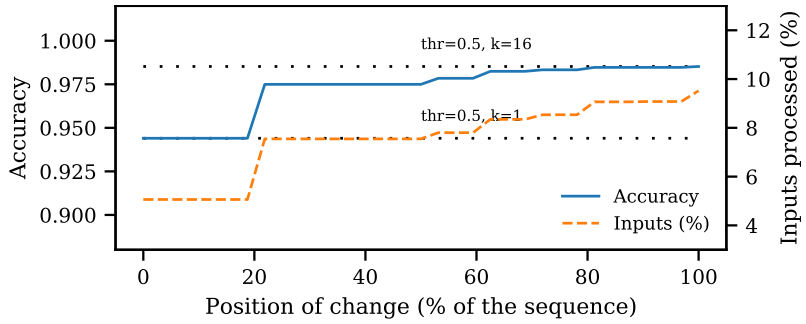

Figure 15: Modifying (thr, K) during the sequence on HAR for SkipW ($\lambda = 2e - 2$, $L = 16$): from tradeoff$_1$: (thr=0.5, K=16, acc/updates = 98.5%/9.5%) to tradeoff$_2$: (thr=0.5, K=1, acc/updates = 94.4%/5.1%).

window and this affects the performance (the model will miss the first marker 80% of the time) but also which inputs are selected for processing in subsequent windows.

D Additional results for HAR

D.1 Modifying tradeoff during inference

Figure 15 and 16 show the adaptability of SkipW during the analysis of a sequence when switching between different tradeoffs. We present here results for a SkipW wih $L = 16$ and $\lambda = 2e - 2$ with different tradeoffs. As with previous experiments (Figure 7 and Figure 10), the performance is bounded by the individual performance of the two configurations evaluated.

E Real time experiments

E.1 Latency experiments

In order to explore the impact of the HW platform on the latency of HAR Services based on SkipW/PoseNet, we measure the latency of the HAR Service on 2 other small platforms: a Jetson TX2 (similar to the Jetson Nano platform (Figure 8), but more powerful) and a Raspberry Pi 4 (better CPU, but lacking a compatible GPU for AI model acceleration). The hardware specification of these different devices is provided in Table 2. Figure 17 compares the performances of the HAR Services on the 3 selected platforms. It is worth noting that the impact of the different tradeoffs ($[K, thr]$) on the latency are very similar.

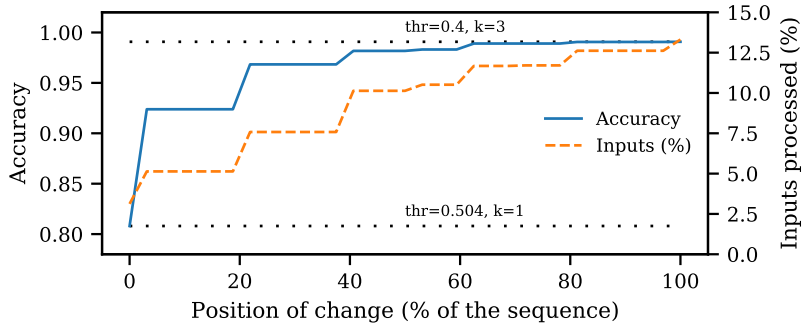

Figure 16: Modifying (thr, K) during the sequence on HAR for SkipW ($\lambda = 2e-2$, $L = 16$): from tradeoff$_1$: (thr=0.4, K=3, acc/updates = 99.1%/13.3%) to tradeoff$_2$: (thr=0.504, K=1, acc/updates = 80.8%/3.1%).

Table 2: Specifications of hardware devices

	Jetson TX2	Jetson Nano	Raspberry Pi 4
AI performance	1.33 TFLOPs	0.47 TFLOPs	0.0135 TFLOPs
GPU[1]	256-core NVIDIA Pascal™	128-core NVIDIA Maxwell™	-
CPU	4-core ARM® A57 (@1.2GHz) & 2-core NVIDIA Denver (@854MHz)	4-core ARM® A57 (@1.2GHz)	4-core ARM® A72 (@1.5GHz)
Memory	4 GB 128-bit LPDDR4 51.2 GB/s	4 GB 64-bit LPDDR4 25.6 GB/s	4 GB LPDDR4-3200

[1] Only AI capable GPUs are reported

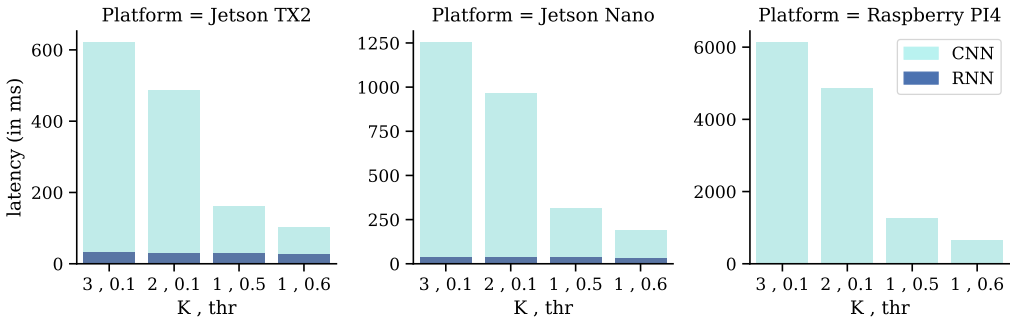

Figure 17: Sequence latency for different devices when processing HAR Service using SkipW ($\lambda = 1e-2$, $L = 8$) coupled with PoseNet (MobileNetV1 0.75). Individual contribution of models (CNN: PoseNet, RNN: SkipW) are also reported.

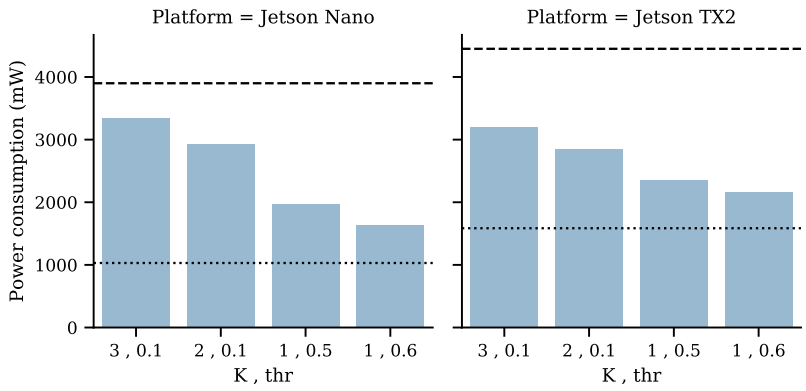

Figure 18: Average energy consumption for the analysis of a HAR sequence using SkipW ($\lambda = 1e-2$, $L = 8$) and PoseNet (MobileNet 0.75) on a Jetson Nano and a Jetson TX2. The dotted line corresponds to the energy level for no activity and the dashed one to the maximum instantaneous level measured when models are running.

E.2 Energy consumption

When SkipW coupled with a CNN model is set appropriately, it can enable a huge reduction in the amount of computation. This can have an immediate impact on the energy consumed to deliver a Service, an important feature for battery powered systems. As an illustration, we measure the power consumed by the HAR analysis when running on the Jetson boards. This analysis is designed to run at 22Hz. The energy consumption is measured by pulling periodically (every 100 ms) the instantaneous power consumption indicator available at the board level. We report here an average of these instantaneous values over a 8 minutes period. Figure 18 provides those energy consumption for the Jetson Nano and the Jetson TX2. Those two platforms display the same behavior: when K decreases, the consumption decreases significantly. The gain is especially noticeable for the Jetson Nano platform, where it can reach up to 50% of energy gain.