Skip RNN: Learning to Skip State Updates in Recurrent Neural Networks

Víctor Campos[*†], **Brendan Jou**[‡], **Xavier Giró-i-Nieto**[§], **Jordi Torres**[†], **Shih-Fu Chang**[Γ]
[†]Barcelona Supercomputing Center, [‡]Google Inc,
[§]Universitat Politècnica de Catalunya, [Γ]Columbia University
{victor.campos, jordi.torres}@bsc.es, bjou@google.com,
xavier.giro@upc.edu, sfchang@ee.columbia.edu

Abstract

Recurrent Neural Networks (RNNs) continue to show outstanding performance in sequence modeling tasks. However, training RNNs on long sequences often face challenges like slow inference, vanishing gradients and difficulty in capturing long term dependencies. In backpropagation through time settings, these issues are tightly coupled with the large, sequential computational graph resulting from unfolding the RNN in time. We introduce the Skip RNN model which extends existing RNN models by learning to skip state updates and shortens the effective size of the computational graph. This model can also be encouraged to perform fewer state updates through a budget constraint. We evaluate the proposed model on various tasks and show how it can reduce the number of required RNN updates while preserving, and sometimes even improving, the performance of the baseline RNN models. Source code is publicly available at https://imatge-upc.github.io/skiprnn-2017-telecombcn/.

1 Introduction

Recurrent Neural Networks (RNNs) have become the standard approach for practitioners when addressing machine learning tasks involving sequential data. Such success has been enabled by the appearance of larger datasets, more powerful computing resources and improved architectures and training algorithms. Gated units, such as the Long Short-Term Memory [24] (LSTM) and the Gated Recurrent Unit [11] (GRU), were designed to deal with the vanishing gradients problem commonly found in RNNs [8]. These architectures have become popularized thanks to their impressive results in a variety of tasks such as machine translation [5], language modeling [53] or speech recognition [19].

Some of the main limitations of RNNs are their challenging training and deployment when dealing with long sequences, due to their inherently sequential behaviour. These challenges include throughput degradation, slower convergence during training and memory leakage, even for gated architectures [38]. Sequence shortening techniques, which can be seen as a sort of conditional computation [7, 6, 15] in time, can alleviate these issues. The most common approaches, such as cropping discrete signals or reducing the sampling rate in continuous signals, are heuristics and can be suboptimal. In contrast, we propose a model that is able to learn which samples (i.e. elements in the input sequence) need to be used in order to solve the target task. Consider a video understanding task as an example: scenes with large motion may benefit from high frame rates, whereas only a few frames are needed to capture the semantics of a mostly static scene.

The main contribution of this work is a novel modification for existing RNN architectures that allows them to skip state updates, decreasing the number of sequential operations to be performed, without requiring any additional supervision signal. This model, called Skip RNN, adaptively determines whether the state needs to be updated or copied to the next time step, thereby allow a "skip" in the

[*]Work done while Víctor Campos was a visiting scholar at Columbia University.

computation graph. We show how the network can be encouraged to perform fewer state updates by adding a penalization term during training, allowing us to train models of different target computation budgets. The proposed modification is implemented on top of well known RNN architectures, namely LSTM and GRU, and the resulting models show promising results in a series of sequence modeling tasks. In particular, the proposed Skip RNN architecture is evaluated on five sequence learning problems: an adding task, sine wave frequency discrimination, digit classification, sentiment analysis in movie reviews and action classification in video.

This paper is structured as follows: Section 2 provides an overview of the related work, Section 3 describes the proposed model, experimental evaluation of Skip RNN in a series of sequence modeling tasks is presented in Section 4, and Section 5 summarizes the main results and some potential extensions of this work. Source code is publicly available at `https://imatge-upc.github.io/skiprnn-2017-telecombcn/`.

2 Related work

Conditional computation has been shown to allow gradual increases in model capacity without a proportional increases in computational cost by exploiting certain computation paths for each input [7, 33, 2, 35, 41]. This idea has been extended in the temporal domain, either by learning how many times an input needs to be pondered before moving to the next one [18] or building RNNs whose number of layers depends on the input data [12]. Some works have addressed time-dependent computation in RNNs by updating only a fraction of the hidden states based on the current hidden state and input [26], or following periodic patterns [29, 38]. However, due to the inherently sequential nature of RNNs and the parallel computation capabilities of modern hardware, reducing the size of the matrices involved in the computations performed at each time step does not accelerate inference. The proposed Skip RNN model can be seen as form of conditional computation in time, where the computation associated to the RNN updates may or may not be executed at every time step. This is related to the UPDATE and COPY operations in hierarchical multiscale RNNs [12], but applied to the whole stack of RNN layers at the same time. This difference is key to allowing our approach to skip input samples, effectively reducing sequential computation and shielding the hidden state over longer time lags. Learning whether to update or copy the hidden state through time steps can be seen as a learnable Zoneout mask [30] which is shared between all the units in the hidden state. Similarly, it can be interpretted as an input-dependent recurrent version of stochastic depth [25].

Selecting parts of the input signal is similar in spirit to the hard attention mechanisms that have been applied to image regions [37], where only some patches of the input image are attended in order to generate captions [49] or detect objects [3]. Our model can be understood to generate a hard temporal attention mask on the fly given the previously seen samples, deciding which time steps should be attended and operating on a subset of input samples. Subsampling input sequences has been explored for visual storylines generation [43], although jointly optimizing the RNN weights and the subsampling mechanism is computationally unfeasible and the Expectation Maximization algorithm is used instead. Similar research has been conducted for video analysis tasks, discovering minimally needed evidence for event recognition [9] and training agents that decide which frames need to be observed in order to localize actions in time [50, 46]. Motivated by the advantages of training recurrent models on shorter subsequences, efforts have been conducted towards learning differentiable subsampling mechanisms [40], although the computational complexity of the proposed method precludes its application to long input sequences. In contrast, our proposed method can be trained with backpropagation and does not degrade the complexity of the baseline RNNs.

Accelerating inference in RNNs is difficult due to their inherently sequential nature, leading to the design of Quasi-Recurrent Neural Networks [10], which relax the temporal dependency between consecutive steps. With the goal of speeding up RNN inference, LSTM-Jump [51] augments an LSTM cell with a classification layer that will decide how many steps to jump between RNN updates. Despite its promising results on text tasks, the model needs to be trained with REINFORCE [48], which requires the definition of a reward signal. Determining such reward signal is not trivial and does not necessarily generalize across tasks, e.g. regression and classification tasks may require from different reward signals. Moreover, the number of tokens read between jumps, the maximum jump distance and the number of jumps allowed need to be chosen ahead of time. These hyperparameters define a reduced set of subsequences that the model can sample, instead of allowing the network to learn any arbitrary sampling scheme. Unlike LSTM-Jump, our proposed approach is differentiable,

thus not requiring any modifications to the loss function and simplifying the optimization process, and is not limited to a predefined set of sample selection patterns.

3 Model Description

An RNN takes an input sequence $\mathbf{x} = (x_1, \ldots, x_T)$ and generates a state sequence $\mathbf{s} = (s_1, \ldots, s_T)$ by iteratively applying a parametric state transition model S from $t = 1$ to T:

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

We augment the network with a binary *state update gate*, $u_t \in \{0, 1\}$, selecting whether the state of the RNN will be updated or copied from the previous time step. At every time step t, the probability $\tilde{u}_{t+1} \in [0, 1]$ of performing a state update at $t + 1$ is emitted. The resulting architecture is depicted in Figure 1 and can be characterized as follows:

$$u_t = f_{binarize}(\tilde{u}_t) \tag{2}$$
$$s_t = u_t \cdot S(s_{t-1}, x_t) + (1 - u_t) \cdot s_{t-1} \tag{3}$$
$$\Delta\tilde{u}_t = \sigma(W_p s_t + b_p) \tag{4}$$
$$\tilde{u}_{t+1} = u_t \cdot \Delta\tilde{u}_t + (1 - u_t) \cdot (\tilde{u}_t + \min(\Delta\tilde{u}_t, 1 - \tilde{u}_t)) \tag{5}$$

where σ is the sigmoid function and $f_{binarize} : [0, 1] \to \{0, 1\}$ binarizes the input value. Should the network be composed of several layers, some columns of W_p can be fixed to 0 so that $\Delta\tilde{u}_t$ depends only on the states of a subset of layers (see Section 4.5 for an example with two layers). We implement $f_{binarize}$ as a deterministic step function $u_t = \text{round}(\tilde{u}_t)$, although a stochastic sampling from a Bernoulli distribution $u_t \sim \text{Bernoulli}(\tilde{u}_t)$ would be possible as well.

The model formulation implements the observation that the likelihood of requesting a new input increases with the number of consecutively skipped samples. Whenever a state update is omitted, the pre-activation of the state update gate for the following time step, $\tilde{u}_{t+1}$, is incremented by $\Delta\tilde{u}_t$. On the other hand, if a state update is performed, the accumulated value is flushed and $\tilde{u}_{t+1} = \Delta\tilde{u}_t$.

The number of skipped time steps can be computed ahead of time. For the particular formulation used in this work, where $f_{binarize}$ is implemented by means of a rounding function, the number of skipped samples after performing a state update at time step t is given by:

$$N_{skip}(t) = \min\{n : n \cdot \Delta\tilde{u}_t \geq 0.5\} - 1 \tag{6}$$

where $n \in \mathbb{Z}^+$. This enables more efficient implementations where no computation at all is performed whenever $u_t = 0$. These computational savings are possible because $\Delta\tilde{u}_t = \sigma(W_p s_t + b_p) = \sigma(W_p s_{t-1} + b_p) = \Delta\tilde{u}_{t-1}$ when $u_t = 0$ and there is no need to evaluate it again, as depicted in Figure 1d.

There are several advantages in reducing the number of RNN updates. From the computational standpoint, fewer updates translates into fewer required sequential operations to process an input signal, leading to faster inference and reduced energy consumption. Unlike some other models that aim to reduce the average number of operations per step [38, 26], ours enables skipping steps completely. Replacing RNN updates with copy operations increases the memory of the network and its ability to model long term dependencies even for gated units, since the exponential memory decay observed in LSTM and GRU [38] is alleviated. During training, gradients are propagated through fewer updating time steps, providing faster convergence in some tasks involving long sequences. Moreover, the proposed model is orthogonal to recent advances in RNNs and could be used in conjunction with such techniques, e.g. normalization [13, 4], regularization [53, 30], variable computation [26, 38] or even external memory [20, 47].

3.1 Error gradients

The whole model is differentiable except for $f_{binarize}$, which outputs binary values. A common method for optimizing functions involving discrete variables is REINFORCE [48], although several estimators have been proposed for the particular case of neurons with binary outputs [7]. We select

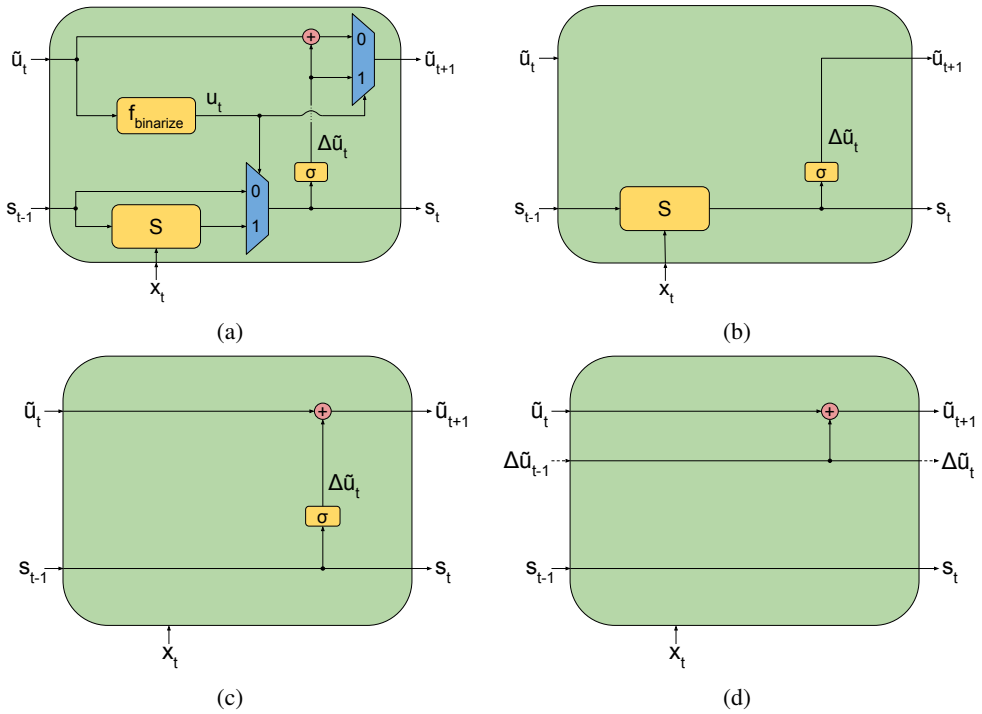

Figure 1: Model architecture of the proposed Skip RNN. **(a)** Complete Skip RNN architecture, where the computation graph at time step t is conditioned on u_t. **(b)** Architecture when the state is updated, i.e. $u_t = 1$. **(c)** Architecture when the update step is skipped and the previous state is copied, i.e. $u_t = 0$. **(d)** In practice, redundant computation is avoided by propagating $\Delta\tilde{u}_t$ between time steps when $u_t = 0$.

the straight-through estimator [23], which consists in approximating the step function by the identity when computing gradients during the backward pass:

$$\frac{\partial f_{binarize}\left(x\right)}{\partial x} = 1 \tag{7}$$

This yields a biased estimator that has proven more efficient than other unbiased but high-variance estimators such as REINFORCE [7] and has been successfully applied in different works [14, 12]. By using the straight-through estimator as the backward pass for $f_{binarize}$, all the model parameters can be trained to minimize the target loss function with standard backpropagation and without defining any additional supervision or reward signal.

3.2 Limiting computation

The Skip RNN is able to learn when to update or copy the state without explicit information about which samples are useful to solve the task at hand. However, a different operating point on the trade-off between performance and number of processed samples may be required depending on the application, e.g. one may be willing to sacrifice a few accuracy points in order to run faster on machines with low computational power, or to reduce energy impact on portable devices. The proposed model can be encouraged to perform fewer state updates through additional loss terms, a common practice in neural networks with dynamically allocated computation [33, 35, 18, 26]. In particular, we consider a *cost per sample*:

$$L_{budget} = \lambda \cdot \sum_{t=1}^{T} u_t \tag{8}$$

where L_{budget} is the cost associated to a single sequence, λ is the cost per sample and T is the sequence length. This formulation bears a similarity to weight decay regularization, where the network is encouraged to slowly converge towards a solution where the norm of the weights is smaller. Similarly, in this case the network is encouraged to slowly converge towards a solution where fewer state updates are required.

Despite this formulation has been extensively studied in our experiments, different budget loss terms can be used depending on the application. For instance, a specific number of samples may be encouraged by applying an L_1 or L_2 loss between the target value and the number of updates per sequence, $\sum_{t=1}^{T} u_t$.

4 Experiments

In the following section, we investigate the advantages of adding this state skipping to LSTMs and GRUs for a variety of tasks. In addition to the evaluation metric for each task, we also report the number of RNN state updates (i.e. the number of elements in the input sequence that are used by the model) as a measure of the computational load for each model. Since skipping an RNN update results in ignoring its corresponding input, we will refer to the number of updates and the number of used samples (i.e. elements in a sequence) interchangeably.

Training is performed with Adam [28], learning rate of 10^{-4}, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$ on batches of 256. Gradient clipping [39] with a threshold of 1 is applied to all trainable variables. Bias b_p in Equation 4 is initialized to 1, so that all samples are used at the beginning of training[2]. The initial hidden state s_0 is learned during training, whereas $\tilde{u}_0$ is set to a constant value of 1 in order to force the first update at $t = 1$.

Experiments are implemented with TensorFlow[3] and run on a single NVIDIA K80 GPU.

4.1 Adding Task

We revisit one of the original LSTM tasks [24], where the network is given a sequence of *(value, marker)* tuples. The desired output is the addition of only the two values that are marked with a 1, whereas those marked with a 0 need to be ignored. We follow the experimental setup by Neil et al. [38], where the first marker is randomly placed among the first 10% of samples (drawn with uniform probability) and the second one is placed among the last half of samples (drawn with uniform probability). This marker distribution yields sequences where at least 40% of the samples are distractors and provide no useful information at all. However, it is worth noting that in this task the risk of missing a marker is very large as compared to the benefits of working on shorter subsequences.

We train RNN models with 110 units each on sequences of length 50, where the values are uniformly drawn from $\mathcal{U}(-0.5, 0.5)$. The final RNN state is fed to a fully connected layer that regresses the scalar output. The model is trained to minimize the Mean Squared Error (MSE) between the output and the ground truth. We consider that a model is able to solve the task when its MSE on a held-out set of examples is at least two orders of magnitude below the variance of the output distribution. This criterion is a stricter version of the one followed in [24].

While all models learn to solve the task, results in Table 1 show that Skip RNN models are able to do so with roughly half of the updates of their corresponding counterparts. Interestingly, Skip LSTM tends to skip more updates than the Skip GRU when no cost per sample is set, behavior that may be related to the lack of output gate in the latter. We hypothesize that there are two possible reasons why the output gate makes the LSTM more prone to skipping updates: (a) it introduces an additional source of memory decay, and (b) it allows to mask out some units in the cell state that may specialize in deciding when to update or copy, making the final regression layer agnostic to such process.

We observed that the models using fewer updates never miss any marker, since the penalization in terms of MSE would be very large (see Figure 2 for examples). These models learn to skip most of the samples in the 40% of the sequence where there are no markers. Moreover, most updates are

[2] In practice, forcing the network to use all samples at the beginning of training improves its robustness against random initializations of its weights and increases the reproducibility of the presented experiments. A similar behavior was observed in other augmented RNN architectures such as Neural Stacks [21].

[3] `https://www.tensorflow.org`

Model	Task solved	State updates
LSTM	Yes	$100.0\% \pm 0.0\%$
Skip LSTM, $\lambda = 0$	Yes	$81.1\% \pm 3.6\%$
Skip LSTM, $\lambda = 10^{-5}$	Yes	$\mathbf{53.9\%} \pm \mathbf{2.1\%}$
GRU	Yes	$100.0\% \pm 0.0\%$
Skip GRU, $\lambda = 0$	Yes	$97.9\% \pm 3.2\%$
Skip GRU, $\lambda = 10^{-5}$	Yes	$\mathbf{50.7\%} \pm \mathbf{2.6\%}$

Table 1: Results for the adding task, displayed as $mean \pm std$ over four different runs. The task is considered to be solved if the MSE is at least two orders of magnitude below the variance of the output distribution.

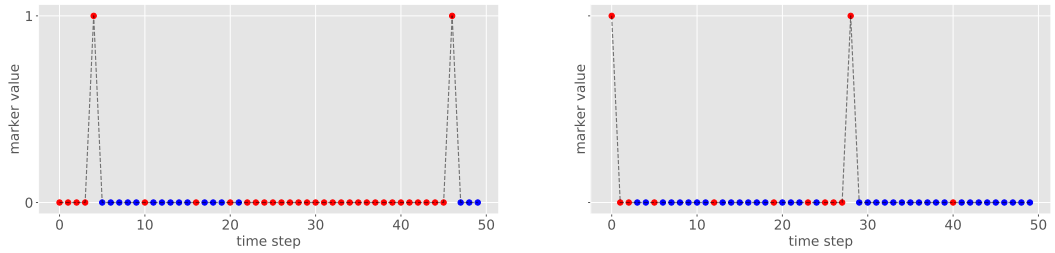

Figure 2: Sample usage examples for the Skip GRU with $\lambda = 10^{-5}$ on the adding task. Red dots indicate used samples, whereas blue ones are skipped.

skipped once the second marker is found, since all the relevant information in the sequence has been already seen. This last pattern provides evidence that the proposed models effectively learn to decide whether to update or copy the hidden state based on the input sequence, as opposed to learning biases in the dataset only. As a downside, Skip RNN models show some difficulties skipping a large number of updates at once, probably due to the cumulative nature of $\tilde{u}_t$.

4.2 Frequency Discrimination Task

In this experiment, the network is trained to classify between sinusoids whose period is in range $T \sim \mathcal{U}(5, 6)$ milliseconds and those whose period is in range $T \sim \{(1, 5) \cup (6, 100)\}$ milliseconds [38]. Every sine wave with period T has a random phase shift drawn from $\mathcal{U}(0, T)$. At every time step, the input to the network is a single scalar representing the amplitude of the signal. Since sinusoid are continuous signals, this tasks allows to study whether Skip RNNs converge to the same solutions when their parameters are fixed but the sampling period is changed. We study two different sampling periods, $T_s = \{0.5, 1\}$ milliseconds, for each set of hyperparameters.

We train RNNs with 110 units each on input signals of 100 milliseconds. Batches are stratified, containing the same number of samples for each class, yielding a 50% chance accuracy. The last state of the RNN is fed into a 2-way classifier and trained with cross-entropy loss. We consider that a model is able to solve the task when it achieves an accuracy over 99% on a held-out set of examples.

Table 2 summarizes results for this task. When no cost per sample is set ($\lambda = 0$), the number of updates differ under different sampling conditions. We attribute this behavior to the potentially large number of local minima in the cost function, since there are numerous subsampling patterns for which the task can be successfully solved and we are not explicitly encouraging the network to converge to a particular solution. On the other hand, when $\lambda > 0$ Skip RNN models with the same cost per sample use roughly the same number of input samples even when the sampling frequency is doubled. This is a desirable property, since solutions are robust to oversampled input signals.

4.3 MNIST Classification from a Sequence of Pixels

The MNIST handwritten digits classification benchmark [32] is traditionally addressed with Convolutional Neural Networks (CNNs) that can efficiently exploit spatial dependencies through weight

Model	$\mathbf{T_s = 1}$**ms (length 100)**		$\mathbf{T_s = 0.5}$**ms (length 200)**	
	Task solved	**State updates**	**Task solved**	**State updates**
LSTM	Yes	100.0 ± 0.00	Yes	200.0 ± 0.00
Skip LSTM, $\lambda = 0$	Yes	55.5 ± 16.9	Yes	147.9 ± 27.0
Skip LSTM, $\lambda = 10^{-5}$	Yes	47.4 ± 14.1	Yes	50.7 ± 16.8
Skip LSTM, $\lambda = 10^{-4}$	Yes	12.7 ± 0.5	Yes	19.9 ± 1.5
GRU	Yes	100.0 ± 0.00	Yes	200.0 ± 0.00
Skip GRU, $\lambda = 0$	Yes	73.7 ± 17.9	Yes	167.0 ± 18.3
Skip GRU, $\lambda = 10^{-5}$	Yes	51.9 ± 10.2	Yes	54.2 ± 4.4
Skip GRU, $\lambda = 10^{-4}$	Yes	23.5 ± 6.2	Yes	22.5 ± 2.1

Table 2: Results for the frequency discrimination task, displayed as $mean \pm std$ over four different runs. The task is considered to be solved if the classification accuracy is over 99%. Models with the same cost per sample ($\lambda > 0$) converge to a similar number of used samples under different sampling conditions.

Model	Accuracy	State updates
LSTM	0.910 ± 0.045	784.00 ± 0.00
Skip LSTM, $\lambda = 10^{-4}$	0.973 ± 0.002	379.38 ± 33.09
GRU	0.968 ± 0.013	784.00 ± 0.00
Skip GRU, $\lambda = 10^{-4}$	0.976 ± 0.003	392.62 ± 26.48

Table 3: Accuracy and used samples on the test set of MNIST after 600 epochs of training. Results are displayed as $mean \pm std$ over four different runs.

sharing. By flattening the 28×28 images into 784-d vectors, however, it can be reformulated as a challenging task for RNNs where long term dependencies need to be leveraged [31]. We follow the standard data split and set aside 5,000 training samples for validation purposes. After processing all pixels with an RNN with 110 units, the last hidden state is fed into a linear classifier predicting the digit class. All models are trained for 600 epochs to minimize cross-entropy loss.

Table 3 summarizes classification results on the test set after 600 epochs of training. Skip RNNs are not only able to solve the task using fewer updates than their counterparts, but also show a lower variation among runs and train faster (see Figure 3). We hypothesize that skipping updates make the Skip RNNs work on shorter subsequences, simplifying the optimization process and allowing the networks to capture long term dependencies more easily. A similar behavior was observed for Phased LSTM, where increasing the sparsity of cell updates accelerates training for very long sequences [38].

Sequences of pixels can be reshaped back into 2D images, allowing to visualize the samples used by the RNNs as a sort of hard visual attention model [49]. Examples such as the ones depicted in Figure 4 show how the model learns to skip pixels that are not discriminative, such as the padding regions in the top and bottom of images. Similarly to the qualitative results for the adding task (Section 4.1), attended samples vary depending on the particular input being given to the network.

4.4 Sentiment Analysis on IMDB

The IMDB dataset [34] contains 25,000 training and 25,000 testing movie reviews annotated into two classes, *positive* and *negative* sentiment, with an approximate average length of 240 words per review. We set aside 15% of training data for validation purposes. Words are embedded into 300-d vector representations before being fed to an RNN with 128 units. The embedding matrix is initialized using pre-trained word2vec[4] embeddings [36] when available, or random vectors drawn from $\mathcal{U}(-0.25, 0.25)$ otherwise [27]. Dropout with rate 0.2 is applied between the last RNN state

[4] `https://code.google.com/archive/p/word2vec/`

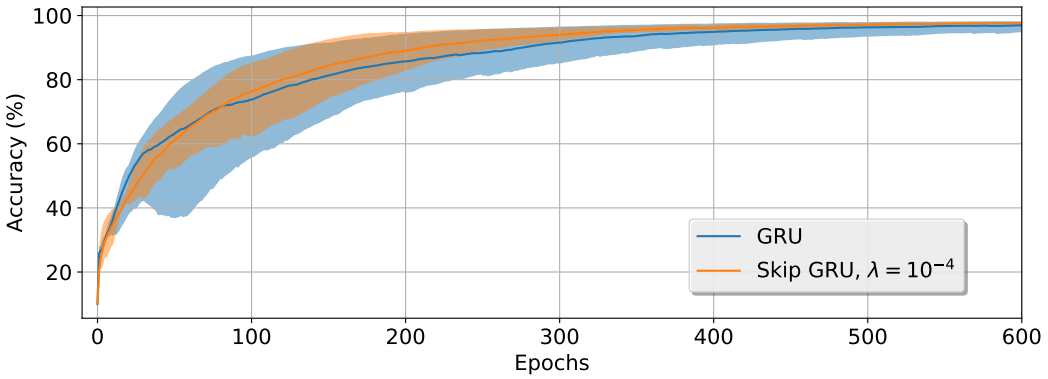

Figure 3: Accuracy evolution during training on the validation set of MNIST. The Skip GRU exhibits lower variance and faster convergence than the baseline GRU. A similar behavior is observed for LSTM and Skip LSTM, but omitted for clarity. Shading shows maximum and minimum over 4 runs, while dark lines indicate the mean.

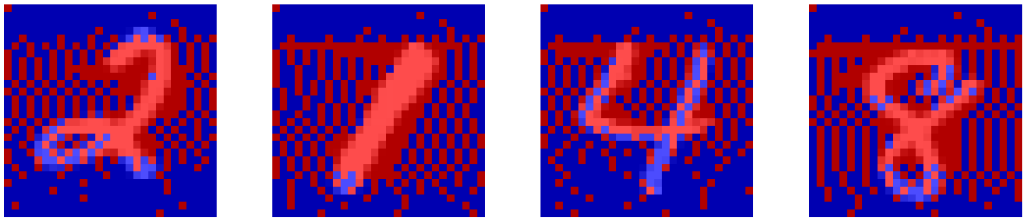

Figure 4: Sample usage examples for the Skip LSTM with $\lambda = 10^{-4}$ on the test set of MNIST. Red pixels are used, whereas blue ones are skipped.

and the classification layer in order to reduce overfitting. We evaluate the models on sequences of length 200 and 400 by cropping longer sequences and padding shorter ones [51].

Results on the test are reported in Table 4. In a task where it is hard to predict which input tokens will be discriminative, the Skip RNN models are able to achieve similar accuracy rates to the baseline models while reducing the number of required updates. These results highlight the trade-off between accuracy and the available computational budget, since a larger cost per sample results in lower accuracies. However, allowing the network to select which samples to use instead of cropping sequences at a given length boosts performance, as observed for the Skip LSTM (length 400, $\lambda = 10^{-4}$), which achieves a higher accuracy than the baseline LSTM (length 200) while seeing roughly the same number of words per review. A similar behavior can be seen for the Skip RNN models with $\lambda = 10^{-3}$, where allowing them to select words from longer reviews boosts classification accuracy while using a comparable number of tokens per sequence.

4.5 Action classification on UCF-101

One of the most accurate and scalable pipelines for video analysis consists in extracting frame level features with a CNN and modeling their temporal evolution with an RNN [17, 52]. Videos are commonly recorded at high sampling rates, rapidly generating long sequences with strong temporal redundancy that are challenging for RNNs. Moreover, processing frames with a CNN is computationally expensive and may become prohibitive for high framerates. These issues have been alleviated in previous works by using short clips [17] or by downsampling the original data in order to cover long temporal spans without increasing the sequence length excessively [52]. Instead of addressing the long sequence problem at the input data level, we train RNN models using long frame sequences without downsampling and let the network learn which frames need to be used.

UCF-101 [44] is a dataset containing 13,320 trimmed videos belonging to 101 different action categories. We use 10 seconds of video sampled at 25fps, cropping longer ones and padding shorter

Model	Length 200		Length 400	
	Accuracy	**State updates**	**Accuracy**	**State updates**
LSTM	0.843 ± 0.003	200.00 ± 0.00	0.868 ± 0.004	400.00 ± 0.00
Skip LSTM, $\lambda = 0$	0.844 ± 0.004	196.75 ± 5.63	0.866 ± 0.004	369.70 ± 19.35
Skip LSTM, $\lambda = 10^{-5}$	0.846 ± 0.004	197.15 ± 3.16	0.865 ± 0.001	380.62 ± 18.20
Skip LSTM, $\lambda = 10^{-4}$	0.837 ± 0.006	164.65 ± 8.67	0.862 ± 0.003	186.30 ± 25.72
Skip LSTM, $\lambda = 10^{-3}$	0.811 ± 0.007	73.85 ± 1.90	0.836 ± 0.007	84.22 ± 1.98
GRU	0.845 ± 0.006	200.00 ± 0.00	0.862 ± 0.003	400.00 ± 0.00
Skip GRU, $\lambda = 0$	0.848 ± 0.002	200.00 ± 0.00	0.866 ± 0.002	399.02 ± 1.69
Skip GRU, $\lambda = 10^{-5}$	0.842 ± 0.005	199.25 ± 1.30	0.862 ± 0.008	398.00 ± 2.06
Skip GRU, $\lambda = 10^{-4}$	0.834 ± 0.006	180.97 ± 8.90	0.853 ± 0.011	314.30 ± 2.82
Skip GRU, $\lambda = 10^{-3}$	0.800 ± 0.007	106.15 ± 37.92	0.814 ± 0.005	99.12 ± 2.69

Table 4: Accuracy and used samples on the test set of IMDB for different sequence lengths. Results are displayed as $mean \pm std$ over four different runs.

Model	Accuracy	State updates
LSTM	0.671	250.0
Skip LSTM, $\lambda = 0$	0.749	138.9
Skip LSTM, $\lambda = 10^{-5}$	0.757	24.2
Skip LSTM, $\lambda = 10^{-4}$	0.790	7.6
GRU	0.791	250.0
Skip GRU, $\lambda = 0$	0.796	124.2
Skip GRU, $\lambda = 10^{-5}$	0.792	29.7
Skip GRU, $\lambda = 10^{-4}$	0.793	23.7

Table 5: Accuracy and used samples on the validation set of UCF-101 (split 1).

examples with empty frames. Activations in the Global Average Pooling layer from a ResNet-50 [22] CNN pretrained on the ImageNet dataset [16] are used as frame level features, which are fed into two stacked RNN layers with 512 units each. The weights in the CNN are not tuned during training to reduce overfitting. The hidden state in the last RNN layer is used to compute the update probability for the Skip RNN models.

We evaluate the different models on the first split of UCF-101 and report results in Table 5. Skip RNN models do not only improve the classification accuracy with respect to the baseline, but require very few updates to do so, possibly due to the low motion between consecutive frames resulting in frame level features with high temporal redundancy [42]. Moreover, Figure 5 shows how models performing fewer updates converge faster thanks to the gradients being preserved during longer spans when training with backpropagation through time.

5 Conclusion

We presented Skip RNNs as an extension to existing recurrent architectures enabling them to skip state updates thereby reducing the number of sequential operations in the computation graph. Unlike other approaches, all parameters in Skip RNN are trained with backpropagation without requiring the introduction of task-dependent hyperparameters like a dropout rate. Experiments conducted with LSTMs and GRUs showed that Skip RNNs can match or in some cases even outperform the baseline models while relaxing their computational requirements. Skip RNNs provide faster and more stable training for long sequences and complex models, likely due to gradients being backpropagated through fewer time steps resulting in a simpler optimization task. Moreover, the introduced computational savings are better suited for modern hardware than those methods that reduce the amount of computation required at each time step [29, 38, 12].

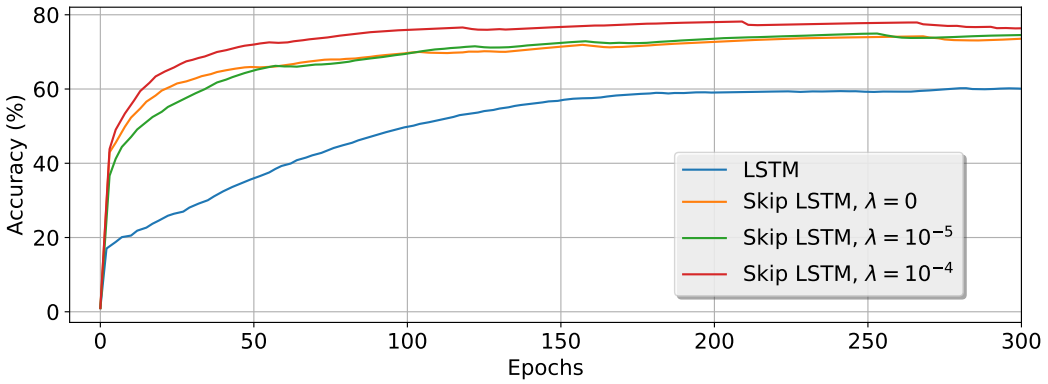

Figure 5: Accuracy evolution during the first 300 training epochs on the validation set of UCF-101 (split 1). Skip LSTM models converge much faster than the baseline LSTM.

The presented results motivate several new research directions toward designing efficient RNN architectures. Introducing stochasticity in neural network training has proven beneficial for generalization [45, 30], and in this work we propose a deterministic rounding operation with stochastic sampling. We showed that the addition of a loss term penalizing the number of updates is important in the performance of Skip RNN and allows flexibility to specialize to tasks of varying budget requirements, e.g. the cost can be increased at each time step to encourage the network to emit a decision earlier [1], or the number of updates can be strictly bounded and enforced. Finally, understanding and analyzing the patterns followed by the model when deciding whether to update or copy the RNN state may provide insight for developing better and more efficient architectures.

Acknowledgments

This work was partially supported by the Spanish Ministry of Economy and Competitivity under contracts TIN2012-34557 by the BSC-CNS Severo Ochoa program (SEV-2011-00067), and contracts TEC2013-43935-R and TEC2016-75976-R. It has also been supported by grants 2014-SGR-1051 and 2014-SGR-1421 by the Government of Catalonia, and the European Regional Development Fund (ERDF). We would also like to thank the technical support team at the Barcelona Supercomputing Center.

References

[1] M. S. Aliakbarian, F. Saleh, M. Salzmann, B. Fernando, L. Petersson, and L. Andersson. Encouraging LSTMs to anticipate actions very early. *arXiv preprint arXiv:1703.07023*, 2017.

[2] A. Almahairi, N. Ballas, T. Cooijmans, Y. Zheng, H. Larochelle, and A. Courville. Dynamic capacity networks. In *ICML*, 2016.

[3] J. Ba, V. Mnih, and K. Kavukcuoglu. Multiple object recognition with visual attention. *arXiv preprint arXiv:1412.7755*, 2014.

[4] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[5] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

[6] Y. Bengio. Deep learning of representations: Looking forward. In *SLSP*, 2013.

[7] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[8] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 1994.

[9] S. Bhattacharya, F. X. Yu, and S.-F. Chang. Minimally needed evidence for complex event recognition in unconstrained videos. In *ICMR*, 2014.

[10] J. Bradbury, S. Merity, C. Xiong, and R. Socher. Quasi-recurrent neural networks. In *ICLR*, 2017.

[11] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*, 2014.

[12] J. Chung, S. Ahn, and Y. Bengio. Hierarchical multiscale recurrent neural networks. In *ICLR*, 2017.

[13] T. Cooijmans, N. Ballas, C. Laurent, Ç. Gülçehre, and A. Courville. Recurrent batch normalization. In *ICLR*, 2017.

[14] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[15] A. Davis and I. Arel. Low-rank approximations for conditional feedforward computation in deep neural networks. *arXiv preprint arXiv:1312.4461*, 2013.

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009.

[17] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *CVPR*, 2015.

[18] A. Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

[19] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *ICASSP*, 2013.

[20] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[21] E. Grefenstette, K. M. Hermann, M. Suleyman, and P. Blunsom. Learning to transduce with unbounded memory. In *NIPS*, 2015.

[22] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[23] G. Hinton. Neural networks for machine learning. Coursera video lectures, 2012.

[24] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 1997.

[25] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016.

[26] Y. Jernite, E. Grave, A. Joulin, and T. Mikolov. Variable computation in recurrent neural networks. In *ICLR*, 2017.

[27] Y. Kim. Convolutional neural networks for sentence classification. In *EMNLP*, 2014.

[28] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[29] J. Koutnik, K. Greff, F. Gomez, and J. Schmidhuber. A clockwork rnn. In *ICML*, 2014.

[30] D. Krueger, T. Maharaj, J. Kramár, M. Pezeshki, N. Ballas, N. R. Ke, A. Goyal, Y. Bengio, H. Larochelle, A. Courville, et al. Zoneout: Regularizing rnns by randomly preserving hidden activations. In *ICLR*, 2017.

[31] Q. V. Le, N. Jaitly, and G. E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

[32] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[33] L. Liu and J. Deng. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. *arXiv preprint arXiv:1701.00299*, 2017.

[34] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *ACL*, 2011.

[35] M. McGill and P. Perona. Deciding how to decide: Dynamic routing in artificial neural networks. In *ICML*, 2017.

[36] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

[37] V. Mnih, N. Heess, A. Graves, et al. Recurrent models of visual attention. In *NIPS*, 2014.

[38] D. Neil, M. Pfeiffer, and S. Liu. Phased LSTM: accelerating recurrent network training for long or event-based sequences. In *NIPS*, 2016.

[39] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *ICML*, 2013.

[40] C. Raffel and D. Lawson. Training a subsampling mechanism in expectation. In *ICLR Workshop Track*, 2017.

[41] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *ICLR*, 2017.

[42] E. Shelhamer, K. Rakelly, J. Hoffman, and T. Darrell. Clockwork convnets for video semantic segmentation. *arXiv preprint arXiv:1608.03609*, 2016.

[43] G. A. Sigurdsson, X. Chen, and A. Gupta. Learning visual storylines with skipping recurrent neural networks. In *ECCV*, 2016.

[44] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[45] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 2014.

[46] Y.-C. Su and K. Grauman. Leaving some stones unturned: dynamic feature prioritization for activity detection in streaming video. In *ECCV*, 2016.

[47] J. Weston, S. Chopra, and A. Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

[48] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 1992.

[49] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, 2015.

[50] S. Yeung, O. Russakovsky, G. Mori, and L. Fei-Fei. End-to-end learning of action detection from frame glimpses in videos. In *CVPR*, 2016.

[51] A. W. Yu, H. Lee, and Q. V. Le. Learning to skim text. In *ACL*, 2017.

[52] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. In *CVPR*, 2015.

[53] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. In *ICLR*, 2015.