Preserving Activations in Recurrent Neural Networks Based on Surprisal

Tayfun Alpay [1] **Fares Abawi** [1] **Stefan Wermter** [1]

Abstract

Learning hierarchical abstractions from sequences is a challenging and open problem for Recurrent Neural Networks (RNNs). This is mainly due to the difficulty of detecting features that span over long time distances with also different frequencies. In this paper, we address this challenge by introducing *surprisal-based activation*, a novel method to preserve activations and skip updates depending on encoding-based information content. The preserved activations can be considered as temporal shortcuts with perfect memory. We present a preliminary analysis by evaluating surprisal-based activation on language modelling with the Penn Treebank corpus and find that it can improve performance when compared to baseline RNNs and Long Short-Term Memory (LSTM) networks.

1. Introduction

Recurrent Neural Networks (RNNs) are powerful sequence processing models that are equipped with memory from recurrent feedback connections. While they are naturally unable to learn long-term dependencies due to vanishing gradients, this issue was successfully addressed by the introduction of activation gating, made popular by the Long Short-Term Memory (LSTM; (Hochreiter & Schmidhuber, 1997)) and Gated Recurrent Units (GRU; (Chung et al., 2014)). The advancement of gating models has led to a wide range of successful practical applications and a generally increased popularity and research on RNNs (Greff et al., 2017).

One of the current main challenges of RNNs is to dynamically adapt to multiple temporal resolutions and scales in order to learn hierarchical representations in time. Since they operate in discrete time steps and update at every time step, it is generally difficult to learn temporal features that have a significantly different resolution than their input frequency. In particular, many applications, such as speech recognition or video analysis, require a high data resolution to capture very short but important events (Zelnik-Manor & Irani, 2001). However, increasing the input resolution has the negative side effect that task-defining events are then very sparsely distributed over the observed time series while most data points are redundant and mostly irrelevant for the task. This causes computational redundancy in state updates of RNNs, ultimately leading to unnecessarily large computational graphs that are hard and expensive to train with Back-Propagation Through Time (BPTT).

This issue with RNNs is most commonly tackled by avoiding high-resolution raw data as input and instead relying on task-specific time-averaging features or down-sampling where possible. There is a number of alternative approaches that can be categorized under conditional computing (Bengio et al., 2013; Campos et al., 2017) in which state updates are optional and based on conditions. Theoretically, this allows the model to actually learn which data points to encode, discarding the rest, which can ultimately even be utilized to save computation time.

We hypothesize that these approaches can be utilized to train (temporal) feature-learning RNNs more efficiently in an end-to-end manner. The capability to learn *when* to update could open up promising directions for a number of current research problems, such as dealing with event-driven extremely long sequences (e.g. raw audio data) without preprocessing, having adaptive computation times (Graves, 2016), learning multiple timescales and their boundaries (Alpay et al., 2016; Koutník et al., 2014), or integrating sensory data under asynchronous sampling conditions (Neil et al., 2016).

In this paper, we introduce surprisal-based activation, an extension to existing RNNs, which allows the inhibition of state updates based on the surprisal over changes in the latent space over time. Preserving activations allows to store explicit memory for an arbitrarily long time before it is accessed and aims to avoid unnecessary changes in the encoding. For this purpose, we extend our previous work (Alpay et al., 2018) by a deeper investigation of how surprisal-based activation can be integrated with LSTMs and delivering a deeper analysis of our method by visualizing the state internals of our trained networks.

[1]Knowledge Technology, Department of Informatics, University of Hamburg, Germany. Correspondence to: Tayfun Alpay <alpay@informatik.uni-hamburg.de>.

Preprint. Final Article: 10.1016/j.neucom.2018.11.092

This paper is organized as follows. Sec. 2 provides an overview of related work that has influenced our approach. Our method is introduced and explained in Sec. 3. We evaluate and analyse our approach extensively on language modelling in Sec. 4. Sec. 5 concludes our paper.

2. Preserving activations

Some recent approaches focus on the idea of suppressing hidden unit activations under specific conditions. The Clockwork RNN (CWRNN; (Koutník et al., 2014)) has a hidden layer that is partitioned into *modules* which are only activated at specific timesteps. Inactive modules simply preserve their previous hidden activations until they are triggered to activate again. These activation triggers are under periodic cycles, static, and determined empirically. It can be shown that such training with multiple update resolutions can facilitate learning multiple timescales (Alpay et al., 2016). A key disadvantage of the CWRNN, however, is that the periodic activation conditions are i) predefined and not learned, and ii) global for the entire sequence. This can lead to challenges when dealing with varying temporal distances between dependencies or phase shifts which are common in real-world applications. This idea has been developed further as a regularization method for RNNs with zoneout (Krueger et al., 2017). Zoneout randomly preserves the previous activations of hidden units and can therefore be seen as a variant of dropout in which connections are masked with a random mask of ones (copy) instead of zeros (drop). In the context of recurrent architectures, zoneout is also a special case of the CWRNN with a single module and clocking timings that are sampled randomly at each timestep.

Skipping irrelevant information has also been achieved with RNNs trained with reinforcement learning (Yu et al., 2017; Johansen & Socher, 2017) although there are successful strategies to estimate gradients for conditional computation (Bengio et al., 2013). Adaptively learning computational boundaries has been achieved with gradient descent using computation penalties (Graves, 2016), binary boundary gates (Chung et al., 2016), and time gates based on rhythmic oscillations (Neil et al., 2016). The Skip RNN even uses binary update gates (Campos et al., 2017) which is similar to our approach, even though our update model enforces an information-theoretic constraint based on surprisal.

Surprisal of the error signal has previously been proposed for a form of adaptive zoneout (Rocki et al., 2016). However, while this is fully adaptive, it depends on additional supervised information in both training and test phases. An important feature of surprisal is that it works well for segmentation (Griffiths et al., 2015) which we utilize to detect information boundaries and preserve activations.

3. Surprisal-based activation

Surprisal is an information-theoretic metric that measures the amount of information conveyed by a particular event. It is inversely related to probability so that improbable events carry more information than probable ones. Surprisal (information content $\mathcal{I}$) is formally defined as:

$$\mathcal{I}(X) = \log\left(\frac{1}{P(X)}\right) = -\log(P(X)) \tag{1}$$

$\mathcal{I}(X)$ is therefore bounded between 0 and the number of bits to store X. Unlikely events X lead to a large surprisal whereas highly probable, in our context *redundant*, events lead to a low surprisal. We utilize these properties of surprisal to implicitly segment the state update computations along boundaries of high surprisal and activity. For this purpose, we redefine any event X as the currently calculated state $\mathbf{h}_t$ at timestep t which we seek to segment according to its activations.

In the following, we assume a single hidden layer $\mathbf{h}_t$ of a standard RNN (recurrent and fully-connected) which naturally generalizes to a multi-layer network. The standard RNN candidate activation $\hat{\mathbf{h}}_t$ is calculated as follows:

$$\hat{\mathbf{h}}_t = f(\mathbf{W}_{xh} \cdot x_t + \mathbf{W}_{hh} \cdot \mathbf{h}_{t-1}), \tag{2}$$

where $\mathbf{W}_{xh}$, $\mathbf{W}_{hh}$ are the input and recurrent weight matrices, and $\mathbf{x}_t$, $\mathbf{h}_{t-1}$ the input and previous state, respectively. Our aim is to now observe $\hat{\mathbf{h}}_t$ and to retroactively determine which of the contained activations to "rewind" to the *previous* timestep $t-1$, thus preserving them for an additional timestep. Consequently, this determines which activations to keep from the *current* timestep, passing them to the final layer output $\mathbf{h}_t$.

Fig. 1 illustrates all computation steps of our approach. We start by partitioning our hidden layer into M even-sized[1] modules $\mathbf{m}_t^{(i)}$ such that $\hat{\mathbf{h}}_t = [\hat{\mathbf{m}}_t^{(1)}, \ldots, \hat{\mathbf{m}}_t^{(M)}]$. The purpose of these modules is to introduce stability through majority weighting since the decision to apply the candidate activation $\hat{\mathbf{h}}_t$ or to preserve $\mathbf{h}_{t-1}$ is made independently for each module. We therefore pool the candidate activations in the next step to compress the activations on a per-module-basis, resulting in the pooling vector $\mathbf{p}_t$ with $|\mathbf{p}_t| = M$:

$$\mathbf{p}_t = g(\hat{\mathbf{h}}_t) = [g(\hat{\mathbf{m}}_t^{(1)}), \ldots, g(\hat{\mathbf{m}}_t^{(M)})], \tag{3}$$

where $g(\cdot)$ is a pooling operator such as max or avg, applied on each module such that each module is projected to

[1]For a simplified implementation and to avoid further hyperparameters, we defined modules to be even-sized. Choices of M are therefore restricted by the constraint $0 \equiv |h| \mod M$. Therefore, having more modules leads to less units per module and vice versa.

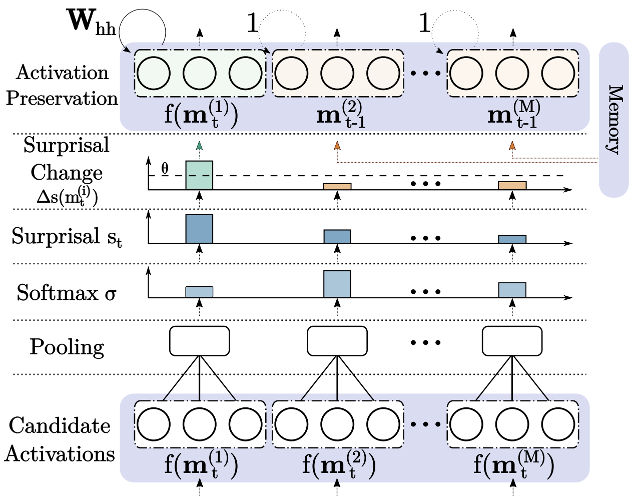


Figure 1. Overview of surprisal-based activation within a hidden layer. Module candidate activations in t that are significantly different to the previous timestep $t-1$ are kept (green), all other modules preserve their previous states $\mathbf{m}_{t-1}^{(i)}$ (red).

a single element with $g(\hat{\mathbf{m}}_t^{(i)}) = \mathbf{p}_t^{(i)}$ while the complete pooling vector $\mathbf{p}_t$, which consists of all of these elements, represents the entirety of all module projections, i.e. the locally pooled hidden layer. In the next step, we normalize with softmax σ and calculate the surprisal s_t from the resulting probability distribution:

$$s_t = \mathcal{I}(\mathbf{p}_t) = \log\left(\frac{1}{\sigma(\mathbf{p}_t)}\right) = \log\left(\frac{\sum_{i=1}^{M}\exp(\mathbf{p}_t^{(i)})}{\exp(\mathbf{p}_t)}\right) \tag{4}$$

As a final step, each module $\mathbf{m}_t^{(i)}$ is activated depending on their respective change in surprisal. We choose candidate activations where the surprisal grows larger than an experimentally determined hyperparameter θ, and otherwise preserve all states of the i-th module:

$$\mathbf{m}_t^{(i)} = \mathcal{S}(\hat{\mathbf{m}}_t^{(i)}) = \begin{cases} \hat{\mathbf{m}}_t^{(i)} & \text{if } s_t > s_{t-1} + \theta, \\ \mathbf{m}_{t-1}^{(i)} & \text{otherwise,} \end{cases} \tag{5}$$

where $\mathcal{S}(\cdot)$ denotes the module-wise transformation of the activations, which is ultimately applied on the candidate activations $\hat{\mathbf{h}}_t$ so that we end up with the final hidden layer output $\mathbf{h}_t$:

$$\mathbf{h}_t = \mathcal{S}(\hat{\mathbf{h}}_t) = [\mathcal{S}(\hat{\mathbf{m}}_t^{(1)}), \ldots, \mathcal{S}(\hat{\mathbf{m}}_t^{(M)})] \tag{6}$$

As a consequence, some modules end up being preserved by $\mathcal{S}(\cdot)$ for this final state $\mathbf{h}_t$ (depending on the condition met in Eq. 5), while others use the original candidate activations, updating their internal model for the current timestep. We henceforth call the resulting model RNN+S (Fig. 1).

All other parts of the RNN are trained normally and backpropagation through time is executed without any modifications.

For a simple interpretation of our model, it is possible to look at it without the information-theoretic motivation which is expressed in the definition of surprisal $\mathcal{I}$ (Eq. 1). The application of surprisal in Eq. 4 causes the pooled activations to be projected to negative log space. This both rescales the activations and introduces a lower bound at 0. While sequential activation comparisons are possible in the original activation space, the negative log space is much more feasible for numerical comparisons as the *difference* can now be expressed regardless of the activation scale. For example, $\mathcal{I}(0.1) - \mathcal{I}(0.2) = \mathcal{I}(0.0001) - \mathcal{I}(0.0002)$, whereas omitting the projection leads to a very large difference between both sides of the equation. One benefit of this is that it allows us to look at this difference and set a threshold θ that is mostly disconnected from the scale of our activations. Since we additionally normalize via softmax before projecting the pooled activations (compare Eq. 4), this leads to an overall more predictable numerical range in which we try to compare activations. From this perspective, our approach can be seen as segmenting activations according to their change in values.

We hypothesize that the combination of local pooling and preservation of previous states based on surprisal maintains the regularizing effect of zoneout, since surprisal-based activation is specifically designed to ignore small perturbations to hidden states, which we would expect to regularize transition dynamics. Another hypothesis is that our design can also lead to a self-organization process in which modules separately learn independent features over long time distances during which the input, and subsequently the encoding, does not undergo significant changes.

3.1. Activation decay

One potential pitfall of preserving activations by internal surprisal is a "deadlock" in which an encoding sequence with mostly constant surprisal is never updated. To counteract this, we investigate the effect of activation decay. We apply decay right before Eq. 5 on the preserved state h_{t-1} and define this decayed state to be $\acute{h}_t$. This changes the negative condition of Eq. 5 to preserving the decayed version of the state:

$$\mathbf{m}_t^{(i)} = \mathcal{S}(\hat{\mathbf{m}}_t^{(i)}) = \begin{cases} \hat{\mathbf{m}}_t^{(i)} & \text{if } s_t > s_{t-1} + \theta, \\ \acute{\mathbf{m}}_{t-1}^{(i)} & \text{otherwise,} \end{cases} \tag{7}$$

Our first approach is to apply a constant decay $d = 1 - \alpha$ $(0 < \alpha < 1)$ to the preservation state $\hat{\mathbf{h}}_{t-1}$ such that $\acute{\mathbf{h}}_t = \mathbf{h}_{t-1} \cdot (1-d)$ constantly decays by the amount α. As a second variant, we introduce a more local and random method by element-wise multiplying the hidden units with a decay mask:

$$\acute{\mathbf{h}}_t = \mathbf{h}_{t-1} \odot \mathbf{d}_t \text{ with } \mathbf{d}_t = \left(P(\mathbf{d}_t^{(1)}), \cdots, P(\mathbf{d}_t^{(|\mathbf{h}|)})\right) \quad (8)$$

where $\mathbf{d}_t$ is a decay vector, and $P(d_j = 1 - \alpha) = p_d$ and $P(d_j = 1) = (1 - p_d)$ are probabilities for decaying the activation of unit j by α with probability p_d. We hypothesize that this introduces variability on a sub-module level and allows single units to counteract potential information loss from pooling. We set these values to $\alpha = 0.01$ and $p_d = 0.2$ as the result of initial tests which indicated that these parameters are not very sensitive as long as they remain small. Since our initial assumption is that deadlocks are global (otherwise other interconnected units would break locally "dead" units) and that few units have to be perturbed to reactivate the entire system, a small probability p_d of causing these perturbations fits this intention. It is however important to note that α is dependant on the range of the used activation function $f(\cdot)$. The same holds true for the threshold θ even though it operates in the log space of the activations. Unbounded activation functions (such as linear or relu activation) would result in different numerical properties and therefore a potentially different set of values. In this paper, our conclusions are specifically based on working with the bounded $\tanh$ and sigmoid activation functions.

3.2. Surprisal-based activation in the LSTM

As the LSTM has more parameters than a standard RNN, there are more possibilities to apply surprisal-based activation. The LSTM has two state variables, i.e. the cell state $\mathbf{c}_t$ and the hidden state $\mathbf{h}_t$. It also has three gates, namely the forget gate $\mathbf{f}_t$, output gate $\mathbf{o}_t$, and input gate $\mathbf{i}_t$. Their interplay, as defined by Greff et al. (Greff et al., 2017), is as follows (omitting bias for clarity):

$$\begin{aligned}
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \hat{\mathbf{c}}_t && (9)\\
\hat{\mathbf{c}}_t &= f(W_{xc} \cdot \mathbf{x}_t + W_{hc} \cdot \mathbf{h}_{t-1}) && (10)\\
\mathbf{h}_t &= \mathbf{o}_t \odot f(\mathbf{c}_t) && (11)\\
\mathbf{i}_t &= \sigma(W_{xi} \cdot \mathbf{x}_t + W_{hi} \cdot \mathbf{h}_{t-1}) && (12)\\
\mathbf{f}_t &= \sigma(W_{xf} \cdot \mathbf{x}_t + W_{hf} \cdot \mathbf{h}_{t-1}) && (13)\\
\mathbf{o}_t &= \sigma(W_{xo} \cdot \mathbf{x}_t + W_{ho} \cdot \mathbf{h}_{t-1}) && (14)
\end{aligned}$$

The preservation function $\mathcal{S}(\cdot)$ can therefore naturally be applied to either or both of $\mathbf{c}_t$ and $\mathbf{h}_t$. For these cases, we can apply Eq. 5 directly on the LSTM. We call the resulting variations S_c, S_h, and S_{ch}.

Since the LSTM forget gate presents an alternative model of memory retrieval to our approach ($\mathbf{f}_t^{(i)} = 0$ causes forgetting, $\mathbf{f}_t^{(i)} = 1$ causes remembering), we also investigate tightly integrating both approaches, i.e. forcing an indirect activation preservation by forcing the network to remember through the forget gates. We therefore extend the preservation function $\mathcal{S}(\cdot)$ for the forget gate:

$$\mathbf{f}_t^{(i)} = \mathcal{S}(\mathbf{f}_t^{(i)}, \mathbf{k}_t) := \begin{cases} \hat{\mathbf{f}}_t^{(i)} & \text{if } \mathcal{I}(\mathbf{k}_t) > \mathcal{I}(\mathbf{k}_{t-1}) + \theta, \\ \mathbf{1} & \text{otherwise,} \end{cases} \quad (15)$$

where $\mathcal{I}(\mathbf{k}_t) = s_t$ gives the surprisal s_t based on the observed vector $\mathbf{k}_t$. We investigate $\mathbf{k}_t \in \{\mathbf{h}_t, \mathbf{c}_t, \mathbf{f}_t\}$ to clarify which of these qualify as the best trigger for the forget gate. These three S_{fk} methods working on the forget gate do not actively modify the cell and state but instead lead to $\mathbf{f}_t^{(i)} = 1$ as the preservation condition. The cell update $\mathbf{c}_t$ therefore changes along with $\mathbf{h}_t$ as follows:

$$\begin{aligned}
\mathbf{h}_t &= \mathbf{o}_t \odot f(\mathbf{c}_t) \\
&= \mathbf{o}_t \odot f(\mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \hat{\mathbf{c}}_t) \\
&= \mathbf{o}_t \odot f(\mathbf{c}_{t-1} + \mathbf{i}_t \odot \hat{\mathbf{c}}_t)
\end{aligned} \quad (16)$$

From $\mathbf{c}_t = \mathbf{c}_{t-1} + \mathbf{i}_t \odot \hat{\mathbf{c}}_t$, it follows that the S_{fk} variants differ from S_c (where $\mathbf{c}_t = \mathbf{c}_{t-1}$) by whether the input gate $\mathbf{i}_t$ is frozen along with the cell $\mathbf{c}_t$ or not.

Setting the input gate $\mathbf{i}_t = 0$, gives a particularly interesting variant in which the cell does not receive any new inputs but simply decays with the forget gate (as briefly discussed but not explored in (Krueger et al., 2017)):

$$\begin{aligned}
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \hat{\mathbf{c}}_t \\
&= \mathbf{f}_t \odot \mathbf{c}_{t-1}
\end{aligned} \quad (17)$$

We therefore also explore this variant, calling it S_{ic}, in which we selectively deactivate the input gate to block *all* outside influence on the memory cell, based on the cell:

$$\mathbf{i}_t^{(i)} = \mathcal{S}(\mathbf{i}_t^{(i)}, \mathbf{c}_t) := \begin{cases} \hat{\mathbf{i}}_t^{(i)} & \text{if } \mathcal{I}(\mathbf{c}_t) > \mathcal{I}(\mathbf{c}_{t-1}) + \theta, \\ \mathbf{0} & \text{otherwise.} \end{cases} \quad (18)$$

All of these discussed variants operate locally and preservations are decided for each module. Overall, we evaluate the following seven variants of surprisal-based activation in an LSTM:

- $LSTM + S_h$: Preserving **hidden states** with $\mathcal{S}(\hat{\mathbf{h}}_t)$
- $LSTM + S_c$: Preserving **cell states** with $\mathcal{S}(\hat{\mathbf{c}}_t)$
- $LSTM + S_{ch}$: Preserving **both** cell and hidden states with $\mathcal{S}(\hat{\mathbf{c}}_t)$, $\mathcal{S}(\hat{\mathbf{h}}_t)$
- $LSTM + S_{fh}$: Setting $\mathbf{f}_t = \mathbf{1}$ based on observing the **hidden state** with $\mathcal{S}(\mathbf{f}_t, \mathbf{h}_t)$
- $LSTM + S_{fc}$: Setting $\mathbf{f}_t = \mathbf{1}$ based on observing the **cell state** with $\mathcal{S}(\mathbf{f}_t, \mathbf{c}_t)$

- $LSTM + S_{ff}$: Setting $\mathbf{f}_t = \mathbf{1}$ based on observing the **forget gate** with $\mathcal{S}(\mathbf{f}_t, \mathbf{f}_t)$
- $LSTM + S_{ic}$: Setting $\mathbf{i}_t = \mathbf{0}$ based on observing the **cell state** with $\mathcal{S}(\mathbf{i}_t, \mathbf{c}_t)$

To summarize, these proposed LSTM variants can be grouped into three separate main approaches: i) those that preserve states directly (S_h, S_c, S_{ch}), ii) those that preserve states by forcing the forget gate to remember (S_{ff}, S_{fh}, S_{fc}), and finally iii) blocking all input to the memory cell (S_{ic}). We compare and evaluate these variants experimentally in the following sections. The objective is to find the most successful setup and also to investigate how the introduced core mechanism interacts with different parameters and how this influences the internal dynamics.

4. Evaluation on language modelling

4.1. Experimental Setup

To investigate the effect of surprisal-based activation, we evaluate our model with language modelling. A task in which the recurrent neural networks process the sentences word by word and have to continually predict the next word at each timestep. Such a model can then be used for autocomplete or text generation systems. For this task, we train on the Penn Treebank corpus, a relatively small but well-known benchmark. We utilize the widely-used preprocessed version by Mikolov (Mikolov, 2012) where 930k words are used as training data, 74k words as validation data, and 82k words as test data. The most frequent 10k training words are used as the vocabulary while the remaining words are replaced by an `<unk>` token. The quality of the trained language models is evaluated by perplexity (PPL), i.e. the exponentiated Cross-Entropy loss $H(p, q)$ of a given model prediction q based on the true label distribution p. The Cross-Entropy itself is typically the average per-word log-probability:

$$\begin{aligned} \mathrm{PPL}(q) &= 2^{H(p,q)} \\ &= 2^{-\frac{1}{N}\sum_{i=1}^{N} \log_2 q(w_i)} \end{aligned} \tag{19}$$

The perplexity expresses how many equally likely predictions the model has on average. Therefore, a low perplexity indicates a more accurate language model. As is usual for While many recent studies make heavy use of regularization techniques to combat the corpus' high susceptibility for overfitting and to reach the lowest possible perplexity, we are mainly concerned with evaluating performance gains from our approach in controlled conditions. We therefore run our own RNN and LSTM models as a baseline and apply our method under the same experimental conditions. Avoiding explicit regularization and task-related tuning methods also allows us to investigate how prone our networks are towards overfitting by

Table 1. Single model validation and test perplexity (the lower, the better) of our models (+S) compared against the LSTM and RNN baselines on the Penn Treebank corpus.

Model	Best Val.	Test
RNN	130.4	140.8
RNN+S	131.5	**126.4**
LSTM	123.6	121.5
LSTM+S_{ch}	128.4	125.7
LSTM+S_c	123.1	120.8
LSTM+S_h	123.9	120.4
LSTM+S_{ff}	125.5	124.0
LSTM+S_{fh}	125.2	123.6
LSTM+S_{fc}	122.3	120.2
LSTM+S_{ic}	116.3	**114.4**

themselves. Our hyperparameters for gradient descent follow (where possible) the previous state of the art for this dataset, achieved by Recurrent Highway Networks (Zilly et al., 2017), even though we restrict ourselves to a single medium-sized layer with 1000 hidden units in order to explore a larger hyperparameter space. Ultimately, we run a total of 1440 different network configurations. We test variations for pooling (max vs. average), activation decay (none, probabilistic, constant), number of modules $M \in \{2, 4, 8, 10, 25, 50, 100, 250, 500, 1000\}$, and threshold $\theta \in \{10^{-7}, 10^{-4}, 10^{-3}\}$. The LSTM models use $tanh$ for activation, the RNN models the sigmoid activation function. Standard gradient descent is used for training with a mini-batch size of 20, a segmented sequence length of 35, and the gradient clipped at 10. Additionally, we use tied word embeddings as described in (Press & Wolf, 2017). To avoid overfitting, training stops at 30 epochs for the RNN and 20 for the LSTM variants, or earlier when the validation perplexity stops improving. We record the epoch with the best validation perplexity. After selecting the best models by this validation perplexity, we then run the evaluation on the test set.

4.2. Generalization and comparison to baseline

Tab. 1 shows the overall results comparing our lowest achieved perplexities against the baselines. As can be seen, surprisal-based activation in an RNN (RNN+S) significantly improves the test perplexity when compared to the baseline (RNN) which can be interpreted as an overall better generalization. It is important to note that, using an extensive hyperparameter search, an unregularized RNN language model can be trained to test perplexities as low as 124.7 (Mikolov, 2012), which in turn would also improve our RNN-specific hyperparameters, and consequently also our RNN+S model. From the LSTM variants, S_h, S_{fc}, and S_c show a slightly better albeit comparable performance to

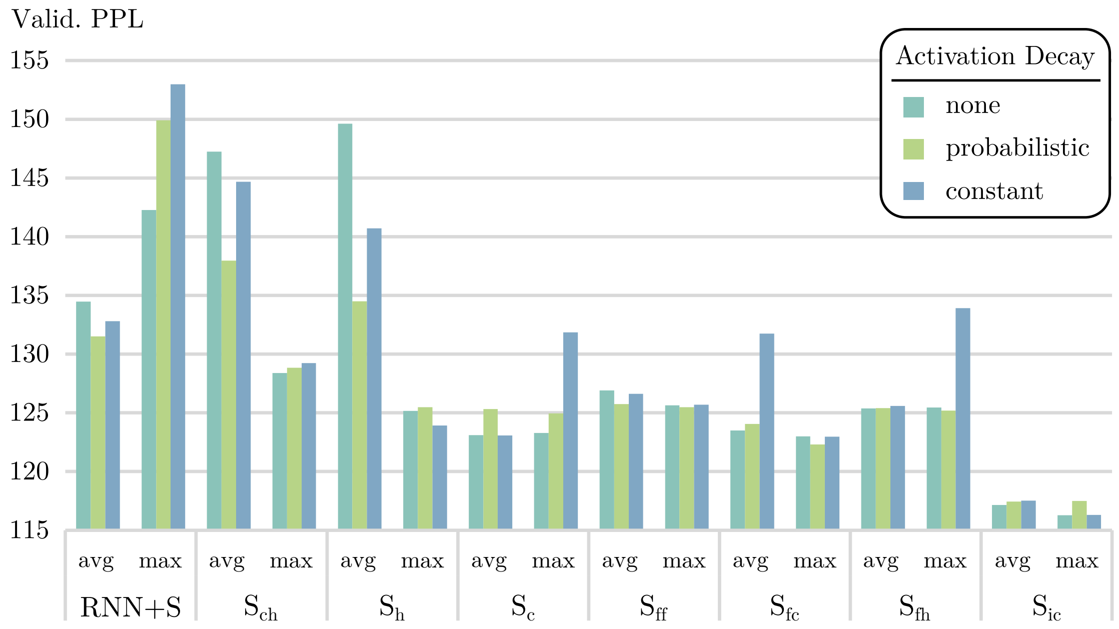


Figure 2. Comparing decay and pooling methods by validation perplexity for all proposed models.

a regular LSTM even though with a slightly better test result. S_{ch}, S_{ff}, and S_{fh} fail to improve the baseline while S_{ic} significantly outperforms the baseline by more than 7 PPL on both validation and test set. In the following sections, all hyperparameter analysis is presented based on the validation set in order to not overfit the test by model selection.

4.3. Pooling and Decay

The impact of pooling and decay can be seen in Fig. 2. Average pooling yields better perplexities for the RNN+S. The negative effect of max pooling is, however, reduced for the LSTM+S, most likely due to a low numerical difference between the average activations in each module and their maximum. Especially in networks with more modules, max pooling has a more positive effect as there is a smaller information loss in pooling smaller modules. Average pooling turns out to be especially detrimental in the LSTM whenever the hidden state is preserved (S_h, S_{ch}), indicating that the activation variance is larger in the hidden state $\mathbf{h}_t$ than the cell $\mathbf{c}_t$. This seems different for the sigmoid-activated gates as all gate-based methods give a similar result for both pooling methods with max pooling yielding a better peak performance.

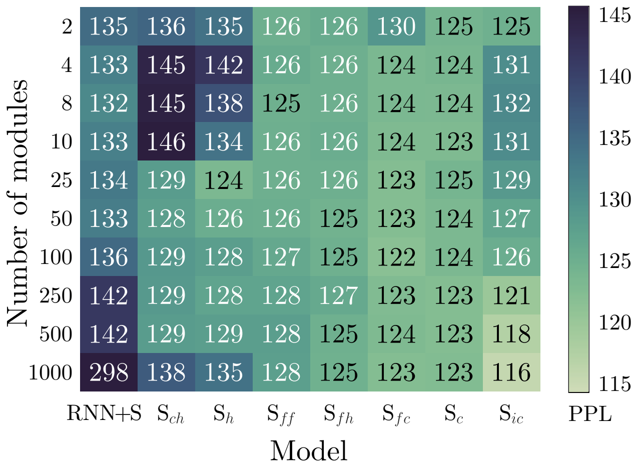
The RNN+S benefits the most from applying a probabilistic unit-level decay mask when using average pooling. It boosts the performance most significantly for S_h and S_{ch} which is in line with our expectation that an initially large activation variance can be toned down by locally applying a decay, lowering the maximum values and improving the average pooling effect. This could be potential evidence that random variations on a lower granularity than the pooling process might help with any surprisal-based "deadlocks" (see Sec. 3), especially when large modules are used which seemingly makes pooling difficult.

Applying the decay globally and at a constant rate has a worse effect for almost all models and shows itself to be an unreliable approach. For a few instances, it is on-par with having no decay at all. One potential explanation for these observations could be that a constant decay at best has no adverse (or beneficial) effect when the activations are converging towards lower values, and at worst, counteracts a change towards larger values. We would particularly expect the approaches based on preserving through the trainable forget gate to not benefit from a constant decay of the gate, as indeed seems to be the case.

Overall, the concept of decay turns out not to be vital for surprisal-based activation in LSTMs as long as the preservation happens through the forget or input gate. The probabilistic decay can potentially serve as a minor performance boost in these cases.

4.4. Number of Modules

Fig. 3 illustrates the effect of the number of modules M on the performance. As can be seen, the LSTM variants S_{ch} and S_h are unable to converge to good solutions for $M < 10$. Since less modules equate to more units per module, the performance drop for $M < 10$ can additionally be explained by the larger area of effect of pooling which



	RNN+S	S_{ch}	S_h	S_{ff}	S_{fh}	S_{fc}	S_c	S_{ic}
2	135	136	135	126	126	130	125	125
4	133	145	142	126	126	124	124	131
8	132	145	138	125	126	124	124	132
10	133	146	134	126	126	124	123	131
25	134	129	124	126	126	123	125	129
50	133	128	126	126	125	123	124	127
100	136	129	128	127	125	122	124	126
250	142	129	128	128	127	123	123	121
500	142	129	129	128	125	124	123	118
1000	298	138	135	128	125	123	123	116

Figure 3. Heatmap showing validation perplexities (PPL) for the RNN+S and all LSTM variants with regard to the module choice M. While S_c gives the most consistent performance for all M, S_{ic} gives the overall best performance for $M \in \{500, 1000\}$ but struggles for smaller choices of M (as do S_{ch}, S_h, and RNN+S).

is consistent with our previous observations in Sec. 4.3 (see Sec. 4.6 for a deeper analysis). However, the opposite end of the spectrum, such as for $M = 1000$ (where each module consists of a single unit) shows that the absence of pooling worsens these two language models (S_{ch} and S_h) considerably. While this effect is even stronger for the RNN+S, the other models do not suffer from this issue. In fact, we can observe an inverse behavior for the best model, the LSTM+S_{ic}, which indicates that it might have quite different dynamics. This model shows itself to be only superior to the other methods for $M \in \{500, 1000\}$. Similar to the state-based preservation methods, its results are quite different for $M < 10$. The LSTM+S_{ic} variant outperforms the others and is best for $M = |h|$. It also has the added benefit that, for this specific configuration, pooling is actually not performed (see Sec. 3) and we ultimately end up with less tunable hyperparameters and an overall simpler model with good performance. S_{ff} and S_c are the only variants which are consistent for all M where the latter gives an overall better result.

4.5. Threshold

The threshold hyperparameter θ ultimately proved to be quite robust to changes for the investigated task. After choosing the best hyperparameter configuration for each model, we have noted the mean average deviations (MAD) of their validation perplexities by looking at different choices of $\theta \in \{10^{-7}, 10^{-4}, 10^{-3}\}$. The results are reported in Tab. 2.

As can be seen, the deviation is largest for the RNN+S, mostly because $\theta = 10^{-7}$ gives better results for $M < 10$

S_{ch}	S_h	S_c	S_{ff}	S_{fc}	S_{fh}	S_{ic}	RNN+S
0.31	0.56	0.11	0.42	0.18	0.32	0.43	4.8

Table 2. Mean absolute deviation (MAD) of validation perplexities with different $\theta \in \{10^{-7}, 10^{-4}, 10^{-3}\}$, given the best configuration per model.

than larger θ configurations. All LSTM variants show a larger robustness towards the choice of θ while the impact is lowest on S_c and highest on S_h. This seems to correlate with their ability to solve the given task. Therefore, the only LSTM configuration where a significant difference can be observed is for the S_h and S_{ch} networks with $M < 10$. As can be seen from Fig. 3, this configuration for M leads to generally bad performance for these two variants, which partly explains this observation.

One potential explanation for the generally low impact of θ is that it is applied within the log-space of the state activations $\mathbf{c}_t$ and $\mathbf{h}_t$ (except for S_{ff} which directly measures the forget gate activation $\mathbf{f}_t$). This means that networks with larger absolute value changes between timesteps are naturally affected less by smaller θ while networks with smaller numerical variance can be forced by larger θ to preserve states more often. Following this assumption, our evaluation shows that our experimental setup operates below this cut-off point, i.e. the chosen θ are small enough to enable learning but also large enough to frequently trigger the activation preservation.

4.6. Further analysis

As discussed in Sec. 3, our motivation for surprisal-based activation originates from the idea to utilize surprisal to temporally segment activations according to their change. Specifically, we look at increasing changes to apply state updates, and block updates for decreasing or low changes. In the context of language modelling, we hypothesize that this improves generalization by learning to preserve important context. This can, for example, happen when the language model is being presented with previously unseen or very rare words. Since wrong predictions can cause a cascade of errors, we want to be able to preserve context and skip timesteps with noisy and wrong predictions.

To investigate our hypothesis, we visualize the internal states. Fig. 4 shows an example sequence which is given to the LSTM+S_{fc}. It can be seen, how measuring the surprisal values s_t on the cell state, helps in segmenting along some boundaries. For example, both **`<eos>`** tokens lead to a change in surprisal before the end of the sentence. The natural change of context that occurs at the sentence boundaries is also slightly reflected in the forget gate where a slightly larger activity can be observed in timestep 8.

A particularly interesting behaviour can be observed be-

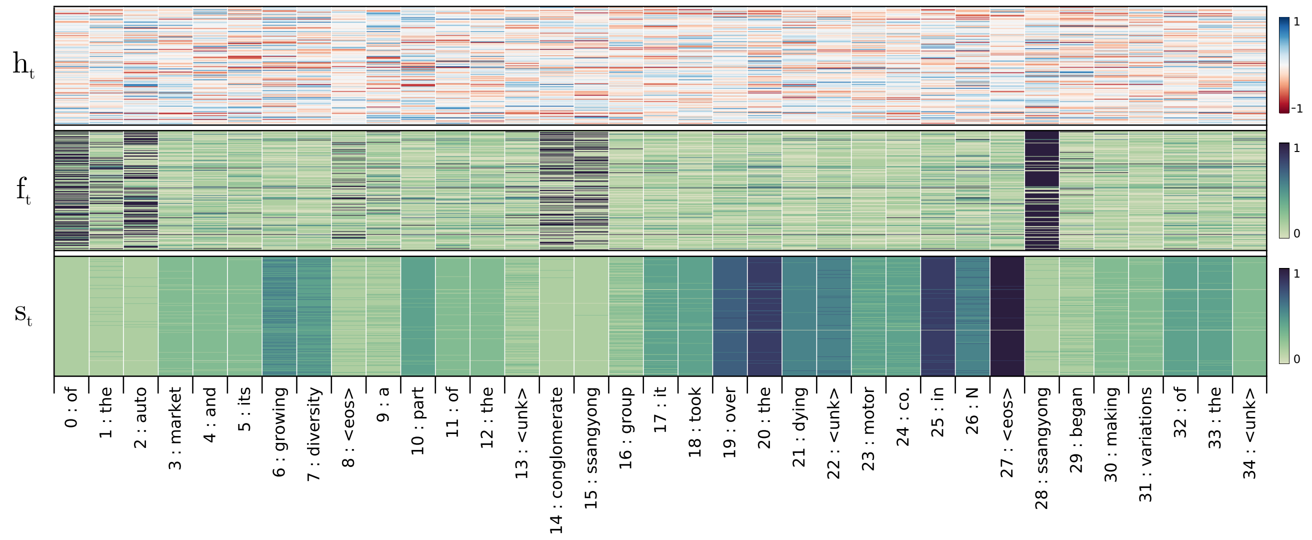


Figure 4. LSTM+S_{fc} with $M = |h|$ processing an example sequence from left to right. The three illustrated maps are from bottom to top (same as the order of calculation): i) the measured surprisal s_t on the cell $\mathbf{c}_t$, ii) the forget gate activations $\mathbf{f}_t$ after applying $\mathcal{S}(\mathbf{f}_t, \mathbf{c}_t)$, and iii) the resulting hidden state activations $\mathbf{h}_t$.

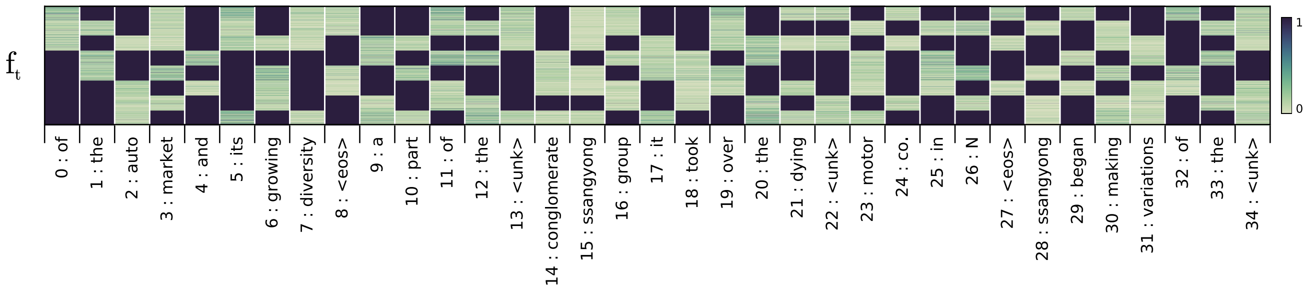


Figure 5. LSTM+S_{fh} with $M = |8|$ processing an example sequence from left to right: For small M, the networks converge to a setup in which preservations are triggered around 50% of the time, resembling standard zoneout.

tween timesteps 13-16 where the input is: **`<unk> conglomerate ssangyong group`**. The surprisal map shows very small values after processing **`conglomerate ssangyong`** which leads to multiple triggered preservations in the forget gate, leading large parts of the network to skip over these rare words. Since only 2 out of 6972 occurrences for the word **`<unk>`** are followed by **`conglomerate`** in the training set, and **`ssangyong`** only occurs once during training, this could in fact be explained by a likely misprediction of this previously unseen context. Since we use a softmax output layer, we believe mispredictions from rare inputs to cause mostly low and widely distributed hidden activations. Very certain predictions, on the other hand, would lead to specific units having high activations due to larger weights. When looking at the hidden activations $\mathbf{h}_t$ (top part of Fig. 4), it is in fact difficult to recognize very strong localized activations, which supports the hypothesis that the network causes mispredictions during these timesteps. Assuming these mispredictions, we can see this behaviour as quite beneficial, as the predictions starting at timestep 16 will have access to the preserved context from timesteps before 13, ignoring its previous mistakes and accessing context that it can utilize.

In order to deepen our understanding, we have also investigated examples that have performed worse than the baseline. As discussed in Sec. 4.4, these networks often have a low number of modules M. For these networks, we have found that surprisal-based activation is sometimes applied seemingly at random. Fig. 5 shows the forget gate activations of the best trained LSTM+S_{fh} network that has $M = 8$ modules (i.e. each module has $1000/8 = 125$ units) and a validation perplexity of 126. It can be seen that preservation is seemingly applied randomly, resembling standard zoneout (Krueger et al., 2017). In fact, we have measured that preservation is applied in 50.11% of the cases for this network. The same network, but with $M = 50$, leads to a significantly lower percentage of preservation with only 6.38%. These findings could potentially explain previous observations where $M < 10$ can lead to worse performance than larger M. It also supports our hypothesis that surprisal-based activation works best when pooling over smaller groups of neurons. For all gate-based variants, setting $M = |h|$ or $M = |h|/2$ to apply preservation on unit-level seems to be the most consistently safe approach. The LSTM+S_{ic} with $M = 250$ has preservations occurring 23.54% of the time. This percentage does, in fact, go up to 38.16% when raising the number of modules to $M = |h| = 1000$, additionally lowering the test perplexity by about 5 points and resulting in our best recorded LSTM+S_{ic} model. From this we can see that frequently triggered preservations do indeed help in generalization when using larger M.

In conclusion, our overall results suggest that, in the context of our experiments, the LSTM cell state is more fit than the hidden state to implement surprisal-based activation. Both the S_{ch} and S_{fh} variants give worse results while operating on $\mathbf{h}_t$ than S_{fc} and S_c, which both observe surprisal changes in the cell $\mathbf{c}_t$. However, measuring and updating only $\mathbf{h}_t$ with the S_h variant, gives a stable performance, indicating that any manipulation of the gates should only be due to changes in the cell states (which is also supported by the worse results for S_{ff}. From our conducted analysis, we can conclude that surprisal-based activation works best by preserving the input gate using the cell state. This model, the $LSTM + S_{ic}$, has also no requirement to fine-tune the pooling method and number of modules M, making it more convenient to use and implement than the other LSTM variants.

5. Conclusion

We have introduced a novel method to preserve activations in RNNs based on surprisal. By investigating and evaluating multiple LSTM integration variants, we have also demonstrated that a baseline LSTM can be improved by surprisal-based blocking and memory decay through the forget and input gates. Especially with the LSTM+S_{ic} we have presented a reliable model when applying surprisal-based activation on unit-level ($M = |h|$), eliminating the necessity to fine-tune most of the introduced hyperparameters.

Our approach provides additional evidence to the overall utility of preserving activations and skipping state updates with neural prediction models. In particular, the presented analysis shows the potential of using surprisal to detect and segment context boundaries. However, further research is needed to conclusively identify whether the detected boundaries do indeed lead to emerging hierarchical structures. Our results are also preliminary since we limited ourselves to a single benchmark to conduct a more extensive analysis. We therefore plan to investigate our approach on other tasks, particularly with much longer sequences, such as in character-level language modelling or question answering.

As demonstrated, surprisal-based activation can be integrated and combined with different types of recurrent layers. In particular, it can be easily extended to GRUs (Chung et al., 2014), which would lead to an additional reduction in complexity.

Future work will also include an investigation into surprisal-based attention as both surprisal and attention are suspected to have related roles for language processing within the predictive coding framework (Zarcone et al., 2016). In general, our approach can be seen as a form of hard attention that is applied at the hidden layer instead of the output layer (Bahdanau et al., 2014; Vinyals et al., 2015).

Acknowledgements

The authors gratefully acknowledge partial support from the German Research Foundation DFG under project CML (TRR 169), the European Union under project SECURE (No 642667), and the NVIDIA Corporation for donating GPUs used in this project. The authors thank Stefan Heinrich for valuable suggestions as well as all reviewers for their helpful remarks.

References

Alpay, Tayfun, Heinrich, Stefan, and Wermter, Stefan. Learning multiple timescales in recurrent neural networks. In *Proceedings of the 25th International Conference on Artificial Neural Networks (ICANN)*, volume 9886 of *Lecture Notes in Computer Science*, pp. 132–139, Barcelona, ES, September 2016. Springer International Publishing. doi: 10.1007/978-3-319-44778-0_16.

Alpay, Tayfun, Abawi, Fares, and Wermter, Stefan. Surprisal-based activation in recurrent neural networks. In *Proceedings of the 26th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, pp. 597–602, Bruges, Belgium, April 2018.

Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. Neural machine translation by jointly learning to align and translate. In *2nd International Conference on Learning Representations (ICLR)*, volume abs/1409.0473, 2014.

Bengio, Yoshua, Léonard, Nicholas, and Courville, Aaron C. Estimating or propagating gradients through stochastic neurons for conditional computation. *Com-*

puting Research Repository (CoRR), abs/1308.3432, 2013.

Campos, Victor, Jou, Brendan, Giró i Nieto, Xavier, Torres, Jordi, and Chang, Shih-Fu. Skip RNN: learning to skip state updates in recurrent neural networks. *Computing Research Repository (CoRR)*, abs/1708.06834, 2017.

Chung, Junyoung, Gulcehre, Caglar, Cho, Kyunghyun, and Bengio, Yoshua. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning (NIPS)*, December 2014.

Chung, Junyoung, Ahn, Sungjin, and Bengio, Yoshua. Hierarchical multiscale recurrent neural networks. In *4th International Conference on Learning Representations (ICLR)*, 2016.

Graves, Alex. Adaptive computation time for recurrent neural networks. *Computing Research Repository (CoRR)*, abs/1603.08983, 2016.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, October 2017. ISSN 2162-237X. doi: 10.1109/TNNLS.2016.2582924.

Griffiths, Sascha S., McGinity, Mariano Mora., Forth, Jamie, Purver, Matthew, and Wiggins, Geraint A. Information-theoretic segmentation of natural language. *International Workshop on Artificial Intelligence and Cognition (AIC)*, pp. 54, 2015.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural Computation (NC)*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735.

Johansen, Alexander Rosenberg and Socher, Richard. Learning when to skim and when to read. *Computing Research Repository (CoRR)*, abs/1712.05483, 2017.

Koutník, Jan, Greff, Klaus, Gomez, Faustino, and Schmidhuber, Jürgen. A clockwork RNN. In *Proceedings of the 31th International Conference on Machine Learning (ICML)*, volume 32, pp. 1863–1871, Bejing, China, June 2014. PMLR.

Krueger, David, Maharaj, Tegan, Kramár, János, Pezeshki, Mohammad, Ballas, Nicolas, Ke, Nan Rosemary, Goyal, Anirudh, Bengio, Yoshua, Courville, Aaron, and Pal, Christopher. Zoneout: Regularizing rnns by randomly preserving hidden activations. In *5th International Conference on Learning Representations (ICLR)*, 2017.

Mikolov, Tomáš. *Statistical language models based on neural networks*. PhD thesis, Brno University of Technology, 2012.

Neil, Daniel, Pfeiffer, Michael, and Liu, Shih-Chii. Phased LSTM: Accelerating recurrent network training for long or event-based sequences. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS)*, pp. 3889–3897, USA, 2016. Curran Associates Inc. ISBN 978-1-5108-3881-9.

Press, Ofir and Wolf, Lior. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL): Volume 2, Short Papers*, pp. 157–163. Association for Computational Linguistics, 2017.

Rocki, Kamil, Kornuta, Tomasz, and Maharaj, Tegan. Surprisal-driven zoneout. *Computing Research Repository (CoRR)*, abs/1610.07675, 2016.

Vinyals, Oriol, Fortunato, Meire, and Jaitly, Navdeep. Pointer networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS)*, volume 2, pp. 2692–2700. MIT Press, 2015.

Yu, Adams Wei, Lee, Hongrae, and Le, Quoc. Learning to skim text. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, volume 1, pp. 1880–1890. Association for Computational Linguistics, 2017. doi: 10.18653/v1/P17-1172.

Zarcone, Alessandra, Van Schijndel, Marten, Vogels, Jorrig, and Demberg, Vera. Salience and attention in surprisal-based accounts of language processing. *Frontiers in Psychology*, 7:844, June 2016.

Zelnik-Manor, L. and Irani, M. Event-based analysis of video. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pp. II–123–II–130 vol.2, 2001. doi: 10.1109/CVPR.2001.990935.

Zilly, Julian Georg, Srivastava, Rupesh Kumar, Koutník, Jan, and Schmidhuber, Jürgen. Recurrent highway networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pp. 4189–4198, Sydney, Australia, August 2017. PMLR.