Université de Montréal

Towards Better Understanding and Improving Optimization in Recurrent Neural Networks

par

Bhargav Kanuparthi

Département d'informatique et de recherche opérationelle
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en Informatique

July 24, 2020

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

Towards Better Understanding and Improving Optimization in Recurrent Neural Networks

présenté par

Bhargav Kanuparthi

a été évalué par un jury composé des personnes suivantes :

Liam Paull
(président-rapporteur)

Yoshua Bengio
(directeur de recherche)

Guillaume Lajoie
(codirecteur)

Aaron Courville
(membre du jury)

Résumé

Les réseaux de neurones récurrents (RNN) sont connus pour leur problème de gradient d'explosion et de disparition notoire (EVGP). Ce problème devient plus évident dans les tâches où les informations nécessaires pour les résoudre correctement existent sur de longues échelles de temps, car il empêche les composants de gradient importants de se propager correctement sur un grand nombre d'étapes. Les articles écrits dans ce travail formalise la propagation du gradient dans les RNN paramétriques et semi-paramétriques pour mieux comprendre la source de ce problème. Le premier article présente un algorithme stochastique simple (h-detach) spécifique à l'optimisation LSTM et visant à résoudre le problème EVGP. En utilisant cela, nous montrons des améliorations significatives par rapport au LSTM vanille en termes de vitesse de convergence, de robustesse au taux d'amorçage et d'apprentissage, et de généralisation sur divers ensembles de données de référence. Le prochain article se concentre sur les RNN semi-paramétriques et les réseaux auto-attentifs. L'auto-attention fournit un moyen par lequel un système peut accéder dynamiquement aux états passés (stockés en mémoire), ce qui aide à atténuer la disparition des gradients. Bien qu'utile, il est difficile à mettre à l'échelle car la taille du graphe de calcul augmente de manière quadratique avec le nombre de pas de temps impliqués. Dans l'article, nous décrivons un mécanisme de criblage de pertinence, inspiré par le processus cognitif de consolidation de la mémoire, qui permet une utilisation évolutive de l'auto-attention clairsemée avec récurrence tout en assurant une bonne propagation du gradient.

Mots clés: Apprentissage automatique, L'apprentissage en profondeur, Réseaux de neurones récurrents, Dépendances à long terme, Problème d'explosion des dégradés de fuite, Réseaux auto-attentifs, Évolutivité.

Abstract

Recurrent neural networks (RNN) are known for their notorious exploding and vanishing gradient problem (EVGP). This problem becomes more evident in tasks where the information needed to correctly solve them exist over long time scales, because it prevents important gradient components from being back-propagated adequately over a large number of steps. The papers written in this work formalizes gradient propagation in parametric and semi-parametric RNNs to gain a better understanding towards the source of this problem. The first paper introduces a simple stochastic algorithm (h-detach) that is specific to LSTM optimization and targeted towards addressing the EVGP problem. Using this we show significant improvements over vanilla LSTM in terms of convergence speed, robustness to seed and learning rate, and generalization on various benchmark datasets. The next paper focuses on semi-parametric RNNs and self-attentive networks. Self-attention provides a way by which a system can dynamically access past states (stored in memory) which helps in mitigating vanishing of gradients. Although useful, it is difficult to scale as the size of the computational graph grows quadratically with the number of time steps involved. In the paper we describe a relevancy screening mechanism, inspired by the cognitive process of memory consolidation, that allows for a scalable use of sparse self-attention with recurrence while ensuring good gradient propagation.

Key words: Machine Learning, Deep Learning, Recurrent Neural Networks, Long Term Dependencies, Exploding Vanishing Gradients Problem, Self Attentive Networks, Scalability.

Contents

List of tables

List of figures

List of Abbreviations

ANN	Artificial neural network
BPC	Bits per character
BPTT	Back propagation through time
CNN	Convolutional neural network
ERM	Empirical risk minimization
EVGP	Exploding vanishing gradients problem
GPU	Graphics processing unit
GRU	Gated recurring unit
IndRNN	Independetly recurrent neural network
iRNN	Identity recurrent neural network
LR	Learning rate

LSTM	Long-short term memory
MLP	Multi Layer Perceptron
orthRNN	Orthogonally initialized recurrent neural network
pMNIST	Permuted MNIST
PTB	Pen treebank
RelLSTM	Relevance long-short term memory
RelRNN	Relevance recurrent neural network
RL	Reinforcement learning
RNN	Recurrent neural network
SAB	Sparse attentive backtracking
SGD	Stochastic gradient descent
uRNN	Unitary recurrent neural network

Acknowledgements

I would like to thank Prof. Yoshua Bengio for his excellent guidance and support throughout the duration of my studies. I would also like to thank Prof. Guillaume Lajoie for his help and supervision in the self-attention project, Dr. Devansh Arpit for his help and contributions to the h-detach project. I would also like to thank Giancarlo Kerg for his contributions and constant support. Finally, I would like to thank Mila for giving me this opportunity and all the members of the lab.

Introduction

With the ever increasing amount of data being generated in today's world the use of Artificial Neural Networks (ANN) has exponentially increased over the past few years. There is an increased need for deep learning [**19**] algorithms to learn better representations of the given data in several domains like Computer Vision, Natural Language Processing (NLP), Speech recognition and Reinforcement learning (RL). These networks use gradient descent based learning methods to learn the optimal solution for a given task.

Recurrent Neural Networks (RNN) [**57**] are a class of neural networks that deal with sequential data. Although useful, these networks become increasingly difficult to train on harder tasks that require learning long term dependencies in the data and have a loss landscape that is much harder to optimize as compared to feed-forward networks. More formally, they run into the Exploding and Vanishing Gradients Problem (EVGP) [**10**]. EVGP is a potential problem for all ANNs but occurs more frequently in RNNs where information has to flow over large time scales. In order to bypass this the use of an attention mechanisms [**5**], [**65**] was proposed with RNNs. The caveat with these mechanisms is that they are memory intensive and are difficult to scale to larger sequences. The work presented in this thesis is a step towards solving the these problems in RNNs. In this chapter we will first lay the foundations required to move forward and describe the previously mentioned problems in detail.

0.1. Introduction to Neural Networks

0.1.1. Artificial Neurons

An **artificial neuron** is a mathematical model inspired by a biological neuron and is the fundamental building block of an ANN. It is connected to and receives several inputs from neighbouring neurons, computes a weighted sum of them and produces a scalar output. This output is then passed through an activation function to get the output of the artificial neuron which in turn can be fed as an input to the next downstream neurons.

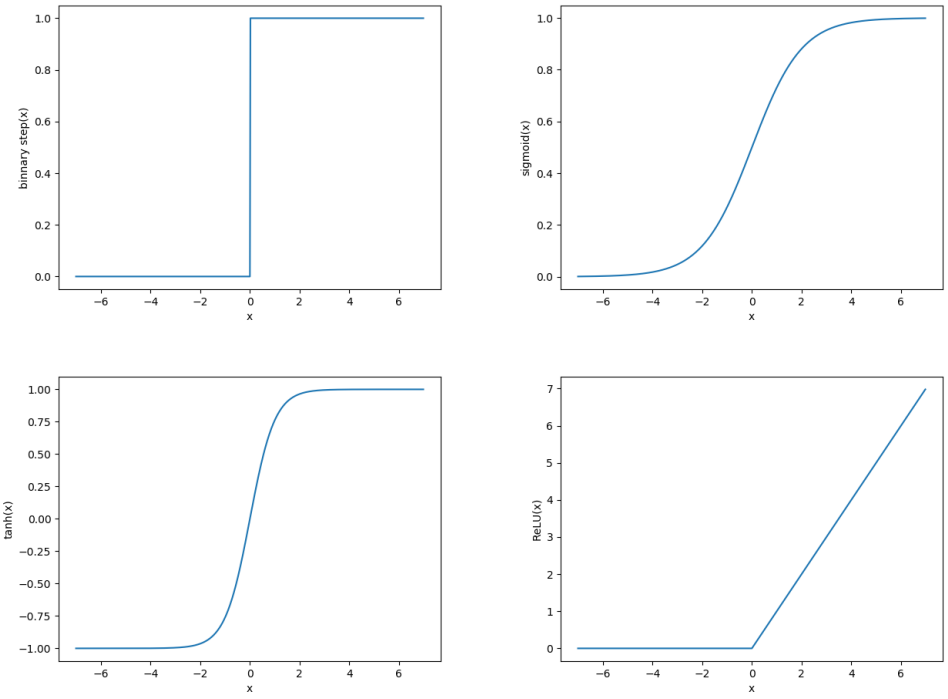

Fig. 1. Graphs of commonly used activation functions. **Binary step** (top left), **Sigmoid** (top right), **tanh** (bottom left) and **ReLU** (bottom right).

More formally, let there be k inputs $\{x_1, x_2,.., x_k\}$ to a neuron and let ϕ be the activation function. Then, we define the function of the artificial neuron $h(x)$ with parameters θ as:

$$h(x) = \phi\left(\sum_{i=1}^{k} w_i x_i + b\right) \tag{0.1.1}$$

where $w_i \in \mathbb{R}$ are the *weights*, $b \in \mathbb{R}$ is the *bias* and $\theta = \{w_1, w_2,..,w_k, b\}$ are the parameters to be learned. The term $\sum_{i=1}^{k} w_i x_i + b$ is referred to as the pre-activation and can also be written as $\mathbf{w}^T \cdot \mathbf{x} + b$ in vector form.

Some commonly used activation functions (Figure 1) are:

- **Binary Step** function - Denoted by $f : \mathbb{R} \to \{0, 1\}$. Defined as,

$$f(x) = \mathbf{1}_{x \geq 0} \tag{0.1.2}$$

- **Sigmoid** function - Denoted by $\sigma : \mathbb{R} \to (0,1)$. Defined as,

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{0.1.3}$$

- **Hyperbolic Tangent** function - Denoted by $\tanh : \mathbb{R} \to (-1, 1)$. Defined as,

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{0.1.4}$$

- **Rectified Linear Unit** function - Denoted by $\text{ReLU} : \mathbb{R} \to [0, \infty)$. Defined as,

$$\text{ReLU}(x) = \max(0, x) = x \cdot \mathbf{1}_{x \geq 0} \tag{0.1.5}$$

0.1.2. Feed-forward Neural Networks

Feed-forward networks or Multi Layer Perceptrons (MLP) are a class of models where several artificial neurons are stacked in a layer-wise structure. It can be defined as having an input layer consisting of d inputs, hidden layers consisting of n_l neurons each and an output layer consisting of k neurons. The flow of information in these networks is only forward, that is information flows from the input layer, through the hidden layers if any and to the output layer.

More formally, suppose we have L hidden layers indexed $l = 1, \ldots, L$ and the neurons of each hidden layer is indexed $j = 1, \ldots, n_j$, then we can write:

$$h_j^{(l)}(\mathbf{x}) = \phi_l \left((\mathbf{w}_j^{(l)})^T \cdot \mathbf{x} + b_j \right) \cdot \tag{0.1.6}$$

where $(\mathbf{w}_j^{(l)}, b_j)$ are the weights and biases respectively for the neuron $h_j^{(l)}$ and ϕ_l is the activation function used for hidden layer l. Note that the size of $\mathbf{x}$ has to be the number of neurons in layer $l - 1$. Another common used notation is to represent the weights for each layer in the form of a matrix $\mathbf{W}^{(l)}$, biases of each layer as a vector $\mathbf{b}^{(l)}$ and the output of each layer as a vector $\mathbf{h}^{(l)}$. Then,

$$\mathbf{h}^{(l)}(\mathbf{x}) = \phi_l \left(\mathbf{W}^{(l)} \cdot \mathbf{x} + \mathbf{b}^{(l)} \right) \cdot \tag{0.1.7}$$

The input layer can be thought of as the layer defined by $l = 0$ and the output layer defined by $l = L$. The overall feed-forward network F can be defined as,

$$F(\mathbf{x}) = \left(\mathbf{h}^{(L)} \circ \mathbf{h}^{(L-1)} \circ \ldots \circ \mathbf{h}^{(1)} \right) (\mathbf{x}) \cdot \tag{0.1.8}$$

The **universal approximation theorem** [**30**] states that a feed-forward network with a single hidden layer and a finite number of neurons can approximate arbitrary well any continuous function on compact sets on $\mathbb{R}^n$, given enough neurons in the hidden layer. The theorem does not describe a way to find the weights, but tells us that simple feed-forward networks can represent a large variety of interesting functions.

0.1.3. Training Artificial Neural Networks

Most ANNs today are trained by gradient based learning methods. Gradient Descent is a first-order optimization algorithm for iteratively finding a local minimum for a given differentiable objective function. Given a dataset $D = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_n}, y_n)\}$ of size n drawn from the underlying data generating distribution p_{data} we would like to adapt the weights and the biases of network to optimize a performance measure P for a given task.

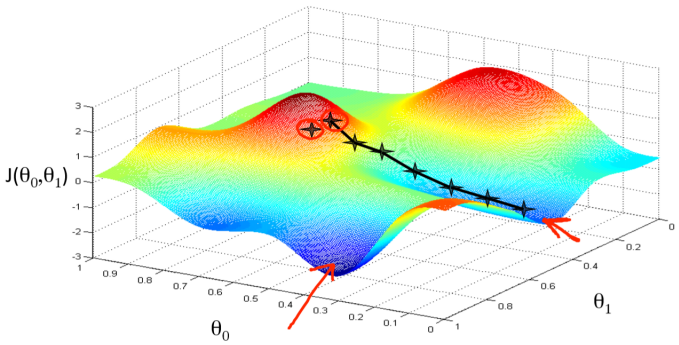

Fig. 2. Illustration of Gradient Descent on an example objective function J with two parameters θ_0 and θ_1. The cross with the red circle denotes the initialization of the network and the goal of the optimization process is to reach the blue areas where the objective function is minimum.

More formally, let θ be the set of network parameters, $F(x;\theta)$ be the network predicted output, y be the target output and $L : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ be the per-example loss function. The objective function we would ideally like to optimize using gradient descent is as follows,

$$J(\theta) = \mathbb{E}_{(\mathbf{x},y)\sim p_{data}}[L(F(\mathbf{x};\theta), y)]\cdot \tag{0.1.9}$$

This objective function is also known as *risk*. However, we do not have access to the data generating distribution p_{data} directly but have access to the empirical distribution $\hat{p}_{data}$ via the dataset D. Hence we can define the objective function $\hat{J}$ as,

$$\hat{J}(\theta) = \mathbb{E}_{(\mathbf{x},y)\sim \hat{p}_{data}}[L(F(\mathbf{x};\theta), y)] = \frac{1}{n}\sum_{i=1}^{n} L(F(\mathbf{x}_i;\theta), y_i)\cdot \tag{0.1.10}$$

This objective function is called *empirical risk* and the process of minimizing it is called Empirical Risk Minimization (ERM). We can now use this objective to compute the gradient in order to update the network. Gradient Descent is an iterative process that takes small steps towards the steepest direction of descent in order to optimize the objective as illustrated in Figure 2. Hence, we can write

$$\theta_{t+1} \leftarrow \theta_t - \eta\nabla_\theta \hat{J}(\theta) \tag{0.1.11}$$

where θ_t is the parameters of the network after t updates and η is the learning rate which controls the size of the step taken.

In practice, Gradient Descent or *batch* Gradient Descent is computationally inefficient as computing $\nabla_\theta \hat{J}(\theta)$ requires the computing the predictions of the entire dataset. A variant known as Stochastic Gradient Descent (SGD) or *mini-batch* Gradient Descent is more commonly used. Instead of computing $\nabla_\theta \hat{J}(\theta)$ over the entire dataset, we compute it over a

subset $\mathcal{S}$ sampled from D where,

$$\mathbb{E}_{\mathcal{S}}[\nabla_\theta \hat{J}_{\mathcal{S}}(\theta)] = \nabla_\theta \hat{J}(\theta) \cdot \tag{0.1.12}$$

In SGD we have $|\mathcal{S}| = 1$ while in *mini-batch* Gradient Descent we have $1 < |\mathcal{S}| \ll n$.

0.1.4. Model Selection and Evaluation

Model selection and evaluation is an important component in training neural networks. In machine learning it is important that the model we select can generalize well to unseen data. Since we optimize $\hat{J}(\theta)$ which is defined over a finite dataset D, picking the model that gives us the lowest $\hat{J}(\theta)$ does not assure us the model will perform well on unseen data. To solve this, we partition $D = D_{\text{train}} \cup D_{\text{valid}}$ and define our objective function $\hat{J}(\theta)$ only based on D_{train}. We split our dataset into a training set and validation set. To evaluate our model we test its performance on D_{valid} and use this as an estimate of its generalization.

The method of picking the best model that gives the best performance on the validation set during training is called *early stopping* since the training is effectively stopped once the validation performance stops improving. Another common way to pick the best model is *k-fold cross validation*. In this D is split into k blocks and the model is repeatedly trained on $k-1$ blocks and tested on the remaining block. The generalization of the model is estimated to be the average of all the test performances.

When it comes to generalization, the two most common problems faced by these models are *overfitting* and *underfitting*. A model is said to overfit the data when its performs well on the training set but poorly on the validation set. On the other hand a model is said to underfit the data when it cannot sufficiently capture the underlying structure of the data. In this case the model performs poorly on the training set and well as the validation set.

0.2. Recurrent Neural Networks

0.2.1. Definition

Recurrent Neural Networks (RNNs) [**57**] are a class of neural networks that deal with sequential data of varying length. They utilize recurrent connections by using the output of the previous time step as an input to the current time step in order to capture temporal information. Also, all the parameters of the RNN are shared across time. They take sequences as input and can output sequences or a scalar value.

More formally, suppose we have an input sequence $x_1, x_2, \ldots, x_T$ where $x_i \in \mathbb{R}^n$, a target output sequence $y_1, y_2, \ldots, y_T$ where $y_i \in \mathbb{R}^k$, then for $t = 1$ to $t = T$, we define the equations

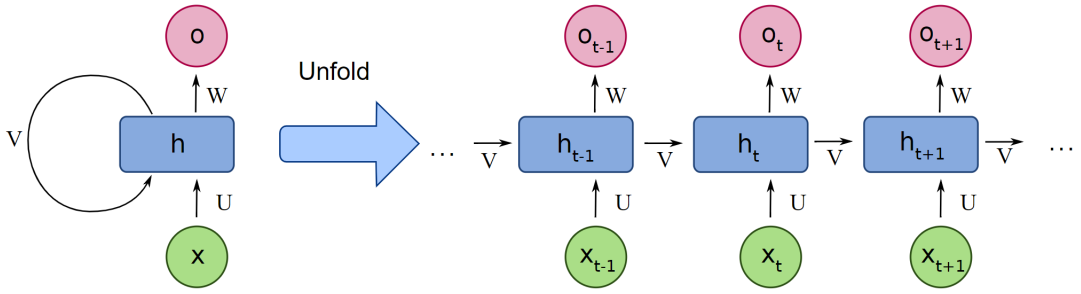

Fig. 3. Illustration of an RNN unrolled in time.

of the RNN as,

$$h_t = \phi(Ux_t + Vh_{t-1} + b) \tag{0.2.1}$$

$$o_t = Wh_t + c \tag{0.2.2}$$

$$\hat{y}_t = \text{softmax}(o_t) \tag{0.2.3}$$

where ϕ is the activation function, h_0 is the initial state, $h_t \in \mathbb{R}^m$ is the hidden state of size m, o_t is the output and $\hat{y}_t$ is the predicted output. $V \in \mathbb{R}^{m\times m}$, $U \in \mathbb{R}^{m\times n}$, $W \in \mathbb{R}^{m\times k}$, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^k$ are the parameters of the network. Let us define $a_t = Ux_t + Vh_{t-1} + b$ as the pre-activation. The **softmax** function: $\mathbb{R}^k \to \mathbb{R}^k$ is defined as,

$$\text{softmax}(z)_j = \frac{e^{z_j}}{\sum_{i=1}^{k} e^{z_k}} \tag{0.2.4}$$

The loss function can L can be written as the sum of the loss function between y_t and $\hat{y}_t$ for all t. That is,

$$L = \sum_t L_t(y_t, \hat{y}_t) \tag{0.2.5}$$

L_t can be a loss function like *mean squared error* or *cross-entropy*. There have been several modifications to RNNs, most notably the use of gating mechanisms like the ones proposed in LSTM [**28**] and GRU [**15**] to control the flow of information over time.

0.2.2. Exploding and Vanishing Gradients Problem in RNNs

In this section we will first derive the gradient of an RNN with respect to the loss function. The forward pass involves unrolling the network over time for a given input sequence of τ time steps. Then a backward pass follows where the gradients are back propagated from $t = \tau$ to $t = 1$. This algorithm is called *back-propagation through time* (BPTT).

Let us assume we have a sequences of length τ and a loss function L is applied only at $t = \tau$. Then, using the RNN equations described in Section 0.2.1 we can derive the gradient

for parameters V, W, W, b and c as,

$$\frac{dL}{dV} = \sum_{t=1}^{\tau} \frac{dL}{da_t} \cdot \frac{\partial a_t}{\partial V} = \sum_{t=1}^{\tau} \frac{dL}{da_t} \cdot h_{t-1}^T = \sum_{t=1}^{\tau} \text{diag}(\phi'(a_t)) \cdot \frac{dL}{dh_t} \cdot h_{t-1}^T \tag{0.2.6}$$

$$\frac{dL}{dU} = \sum_{t=1}^{\tau} \frac{dL}{da_t} \cdot \frac{\partial a_t}{\partial U} = \sum_{t=1}^{\tau} \frac{dL}{da_t} \cdot x_t^T = \sum_{t=1}^{\tau} \text{diag}(\phi'(a_t)) \cdot \frac{dL}{dh_t} \cdot x_t^T \tag{0.2.7}$$

$$\frac{dL}{dW} = \frac{dL}{do_\tau} \cdot \frac{do_\tau}{dW} = \frac{dL}{do_\tau} \cdot h_\tau^T \tag{0.2.8}$$

$$\frac{dL}{db} = \sum_{t=1}^{\tau} \frac{dL}{da_t} \cdot \frac{da_t}{db} = \sum_{t=1}^{\tau} \text{diag}(\phi'(a_t)) \cdot \frac{dL}{dh_t} \tag{0.2.9}$$

$$\frac{dL}{dc} = \frac{dL}{do_\tau} \cdot \frac{do_\tau}{dc} = \frac{dL}{do_\tau} \tag{0.2.10}$$

For the derivation in the case where $L = \sum_t L_t$ the corresponding gradients are summed up. Let us define $D_t = \text{diag}(\phi'(a_t))$. Notice that $\frac{dL}{do_\tau}$ is dependent on the loss function L and is simple to compute. Now we can derive $\frac{dL}{dh_t}$ as,

$$\frac{dL}{dh_t} = \left(\frac{do_\tau}{dh_t}\right)^T \cdot \frac{dL}{do_\tau} = \left(\frac{dh_\tau}{dh_t}\right)^T \cdot W^T \cdot \frac{dL}{do_\tau} \tag{0.2.11}$$

$$\left(\frac{dh_\tau}{dh_t}\right)^T = \prod_{k=t}^{\tau-1} \left(\frac{dh_{k+1}}{dh_k}\right)^T = \prod_{k=t}^{\tau-1} V^T D_{k+1} \tag{0.2.12}$$

$$\frac{dL}{dh_t} = \left(\prod_{k=t}^{\tau-1} V^T D_{k+1}\right) \cdot W^T \cdot \frac{dL}{do_\tau}. \tag{0.2.13}$$

Let us first assume we use identity as the activation function, that is $\phi'(x) = \text{Id}$. Notice that in the above equation the matrix V is present as $(V^T)^{\tau-t}$ which leads to the exploding and vanishing gradients problem (EVGP) [**10**], [**26**] when $t \ll \tau$. The value of $(V^T)^{\tau-t}$ depends on the largest eigenvalue of V. If the largest eigenvalue of V is less than 1 the final matrix would vanish and be close to 0 and if it is greater than 1 the final value would explode and become infinity. If ϕ is not the identity, then that could also be a source for EVGP based on the values of D_{k+1}.

This is a common problem when the network has to learn tasks with long term dependencies, if an event at time t is crucial in determining the outcome at time τ where $t \ll \tau$. Then if $\frac{dL}{dh_t}$ vanishes the parameters will not be updated to learn this dependency while if it explodes it will not contribute to a meaningful gradient for a gradient descent based learning algorithm. Hence, training RNNs on tasks with long term dependencies is difficult.

0.2.3. Self-Attentive Recurrent Networks

Self-Attentive RNNs are networks that have semi-parametric memory modules as opposed to vanilla RNNs that have parametric memory modules. This is because in addition to the

hidden state the network also has access to and attends over all the past hidden states. The concept of attention was originally introduced in [**5**] in which a decoder network attended over all the hidden states of the encoder network. Self-attention is different as the network attends over past hidden states of the same sequence [**65**] and is more formally described in the second article. This introduces skip connections in the sequence and allows the information to directly flow from one time step to any other time step.

Although useful, these networks do not scale as the size of the computational graph increases quadratically with time. The following two articles cover all the related work towards solving these problems faced by RNNs and self-attentive networks respectively, and present ways of mitigating them.

First Article.

h-detach: Modifying the LSTM Gradient Towards Better Optimization

by

Bhargav Kanuparthi[*,1], Devansh Arpit[*,1], Giancarlo Kerg[1],
Nan Rosemary Ke[1], Ioannis Mitliagkas[1], and Yoshua Bengio[1]

([1]) Université de Montréal, Département d'informatique et de recherche opérationelle, Montreal, Canada and Mila - Quebec AI Institute

This article was submitted and published as a conference paper at **ICLR 2019**.

The main contributions of Bhargav Kanuparthi for this article are presented.

- Proposed the initial idea of the h-detach algorithm.
- Implementation of all the experiments for h-detach and hyperparameter searches to find best results on standard benchmark tasks.
- Involved in designing and fully implementing ablation studies to get a deeper understanding of the algorithm.
- Involved in discussions and helped with the proofs of the main theorems.
- Writing of the experimental results and discussion section of the published paper.

Abstract. Recurrent neural networks are known for their notorious exploding and vanishing gradient problem (EVGP). This problem becomes more evident in tasks where the information needed to correctly solve them exist over long time scales, because EVGP prevents important gradient components from being back-propagated adequately over a large number of steps. We introduce a simple stochastic algorithm (h-detach) that is specific to LSTM optimization and targeted towards addressing this problem. Specifically, we show that when the LSTM weights are large, the gradient components through the linear path (cell state) in the LSTM computational graph get suppressed. Based on the hypothesis that these components carry information about long term dependencies (which we show empirically), their suppression can prevent LSTMs from capturing them. Our algorithm prevents gradients flowing through this path from getting suppressed, thus allowing the LSTM to capture such dependencies better. We show significant improvements over vanilla LSTM gradient based training in terms of convergence speed, robustness to seed and learning rate, and generalization using our modification of LSTM gradient on various benchmark datasets.
Keywords: LSTM, Exploding and vanishing gradients problem.

1. Introduction

Recurrent Neural Networks (RNNs) ([**57, 17**]) are a class of neural network architectures used for modeling sequential data. Compared to feed-forward networks, the loss landscape of recurrent neural networks are much harder to optimize. Among others, this difficulty may be attributed to the exploding and vanishing gradient problem [**26, 10, 53**] which is more severe for recurrent networks and arises due to the highly ill-conditioned nature of their loss surface. This problem becomes more evident in tasks where training data has dependencies that exist over long time scales.

Due to the aforementioned optimization difficulty, variants of RNN architectures have been proposed that aim at addressing these problems. The most popular among such architectures that are used in a wide number of applications include long short term memory (LSTM, [**28**]) and gated recurrent unit (GRU, [**15**]) networks, which is a variant of LSTM with forget gates [**18**]. These architectures mitigate such difficulties by introducing a *linear temporal path* that allows gradients to flow more freely across time steps. [**2**] on the other hand try to address this problem by parameterizing a recurrent neural network to have unitary transition matrices based on the idea that unitary matrices have unit singular values which prevents gradients from exploding/vanishing.

Among the aforementioned RNN architectures, LSTMs are arguably most widely used (for instance they have more representational power compared with GRUs [**69**]) and it remains a hard problem to optimize them on tasks that involve long term dependencies. Examples of such tasks are copying problem [**10, 53**], and sequential MNIST [**39**], which are designed in such a way that the only way to produce the correct output is for the model to retain information over long time scales.

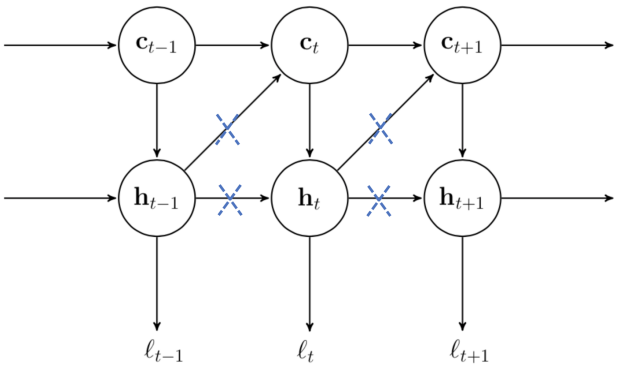

Fig. 4. The computational graph of a typical LSTM. Here we have omitted the inputs $\mathbf{x}_i$ for convenience. The top horizontal path through the cell state units $\mathbf{c}_t$s is the linear temporal path which allows gradients to flow more freely over long durations. The dotted blue crosses along the computational paths denote the stochastic process of blocking the flow of gradients though the $\mathbf{h}_t$ states (see Eq 2.2) during the back-propagation phase of LSTM. We call this approach h-detach.

The goal of this paper is to introduce a simple trick that is specific to LSTM optimization and improves its training on tasks that involve long term dependencies. To achieve this goal, we write out the full back-propagation gradient equation for LSTM parameters and split the composition of this gradient into its components resulting from different paths in the unrolled network. We then show that when LSTM weights are large in magnitude, the gradients through the linear temporal path (cell state) get suppressed (recall that this path was designed to allow smooth gradient flow over many time steps). We show empirical evidence that this path carries information about long term dependencies (see section 3.5) and hence gradients from this path getting suppressed is problematic for such tasks. To fix this problem, we introduce a simple stochastic algorithm that in expectation scales the individual gradient components, which prevents the gradients through the linear temporal path from being suppressed. In essence, the algorithm stochastically prevents gradient from flowing through the h-state of the LSTM (see figure 4), hence we call it h-detach. Using this method, we show improvements in convergence/generalization over vanilla LSTM optimization on the copying task, transfer copying task, sequential and permuted MNIST, and image captioning.

2. Proposed Method: h-detach

We begin by reviewing the LSTM roll-out equations. We then derive the LSTM back-propagation equations and by studying its decomposition, identify the aforementioned problem. Based on this analysis we propose a simple stochastic algorithm to fix this problem.

2.1. Long Short Term Memory Networks

LSTM is a variant of traditional RNNs that was designed with the goal of improving the flow of gradients over many time steps. The roll-out equations of an LSTM are as follows,

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t \tag{2.1}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot tanh(\mathbf{c}_t) \tag{2.2}$$

where $\odot$ denotes point-wise product and the gates $\mathbf{f}_t$, $\mathbf{i}_t$, $\mathbf{o}_t$ and $\mathbf{g}_t$ are defined as,

$$\mathbf{g}_t = tanh(\mathbf{W}_{gh}\mathbf{h}_{t-1} + \mathbf{W}_{gx}\mathbf{x}_t + \mathbf{b}_g) \tag{2.3}$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_{fh}\mathbf{h}_{t-1} + \mathbf{W}_{fx}\mathbf{x}_t + \mathbf{b}_f) \tag{2.4}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_{ih}\mathbf{h}_{t-1} + \mathbf{W}_{ix}\mathbf{x}_t + \mathbf{b}_i) \tag{2.5}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_{oh}\mathbf{h}_{t-1} + \mathbf{W}_{ox}\mathbf{x}_t + \mathbf{b}_o)\cdot \tag{2.6}$$

Here $\mathbf{c}_t$ and $\mathbf{h}_t$ are the cell state and hidden state respectively. Usually a transformation $\phi(\mathbf{h}_T)$ is used as the output at time step t (Eg. next word prediction in language model) based on which we can compute the loss $\ell_t := \ell(\phi(\mathbf{h}_t))$ for that time step.

An important feature of the LSTM architecture is the linear recursive relation between the cell states $\mathbf{c}_t$ as shown in Eq. 2.1. This linear path allows gradients to flow easily over long time scales. This however is one of the components in the full composition of the LSTM gradient. As we will show next, the remaining components that are a result of the other paths in the LSTM computational graph are polynomial in the weight matrices $\mathbf{W}_{gh}$,$\mathbf{W}_{fh}$, $\mathbf{W}_{ih}$, $\mathbf{W}_{oh}$ whose order grows with the number of time steps. These terms cause an imbalance in the order of magnitude of gradients from different paths, thereby suppressing gradients from linear paths of LSTM computational graph in cases where the weight matrices are large.

2.2. Back-propagation Equations for LSTM

In this section we derive the back-propagation equations for LSTM network and by studying its composition, we identify a problem in this composition. The back-propagation equation of an LSTM can be written in the following form.

Theorem 1. *Fix w to be an element of the matrix $\mathbf{W}_{gh}$,$\mathbf{W}_{fh}$, $\mathbf{W}_{ih}$, $\mathbf{W}_{oh}$,$\mathbf{W}_{gx}$,$\mathbf{W}_{fx}$, $\mathbf{W}_{ix}$ or $\mathbf{W}_{ox}$. Define,*

$$\mathbf{A}_t = \begin{bmatrix} \mathbf{F}_t & \mathbf{0}_n & diag(\mathbf{k}_t) \\ \tilde{\mathbf{F}}_t & \mathbf{0}_n & diag(\tilde{\mathbf{k}}_t) \\ \mathbf{0}_n & \mathbf{0}_n & \mathbf{Id}_n \end{bmatrix} \quad \mathbf{B}_t = \begin{bmatrix} \mathbf{0}_n & \psi_t & \mathbf{0}_n \\ \mathbf{0}_n & \tilde{\psi}_t & \mathbf{0}_n \\ \mathbf{0}_n & \mathbf{0}_n & \mathbf{0}_n \end{bmatrix} \quad \mathbf{z}_t = \begin{bmatrix} \frac{d\mathbf{c}_t}{dw} \\ \frac{d\mathbf{h}_t}{dw} \\ \mathbf{1}_n \end{bmatrix} \tag{2.7}$$

Then $\mathbf{z}_t = (\mathbf{A}_t + \mathbf{B}_t)\mathbf{z}_{t-1}$. *In other words,*

$$\mathbf{z}_t = (\mathbf{A}_t + \mathbf{B}_t)(\mathbf{A}_{t-1} + \mathbf{B}_{t-1}) \dots (\mathbf{A}_2 + \mathbf{B}_2)\mathbf{z}_1 \tag{2.8}$$

where all the symbols used to define $\mathbf{A}_t$ *and* $\mathbf{B}_t$ *are defined in notation 1 in appendix.*

To avoid unnecessary details, we use a compressed definitions of $\mathbf{A}_t$ and $\mathbf{B}_t$ in the above statement and write the detailed definitions of the symbols that constitute them in notation 1 in appendix. Nonetheless, we now provide some intuitive properties of the matrices $\mathbf{A}_t$ and $\mathbf{B}_t$.

The matrix $\mathbf{A}_t$ contains components of parameter's full gradient that arise due to the cell state (linear temporal path) described in Eq. (2.1) (top most horizontal path in figure 4). Thus the terms in $\mathbf{A}_t$ are a function of the LSTM gates and hidden and cell states. Note that all the gates and hidden states $\mathbf{h}_t$ are bounded by definition because they are a result of sigmoid or tanh activation functions. The cell state $\mathbf{c}_t$ on the other hand evolves through a linear recursive equation shown in Eq. (2.1). Thus it can grow at each time step by at most ± 1 (element-wise) and its value is bounded by the number of time steps t. Thus given a finite number of time steps and finite initialization of $\mathbf{c}_0$, the values in matrix $\mathbf{A}_t$ are bounded.

The matrix $\mathbf{B}_t$ on the other hand contains components of parameter's full gradient that arise due to the remaining paths. The elements of $\mathbf{B}_t$ are a linear function of the weights $\mathbf{W}_{gh}, \mathbf{W}_{fh}, \mathbf{W}_{ih}, \mathbf{W}_{oh}$. Thus the magnitude of elements in $\mathbf{B}_t$ can become very large irrespective of the number of time steps if the weights are very large. This problem becomes worse when we multiply $\mathbf{B}_t$s in Eq. (2.8) because the product becomes polynomial in the weights which can become unbounded for large weights very quickly as the number of time steps grow.

Thus based on the above analysis, we identify the following problem with the LSTM gradient: when the LSTM weights are large, the gradient component through the cell state paths ($\mathbf{A}_t$) get suppressed compared to the gradient components through the other paths ($\mathbf{B}_t$) due to an imbalance in gradient component magnitudes. We recall that the linear recursion in the cell state path was introduced in the LSTM architecture [**28**] as an important feature to allow gradients to flow smoothly through time. As we show in our ablation studies (section 3.5), this path carries information about long term dependencies in the data. Hence it is problematic if the gradient components from this path get suppressed.

2.3. *h*-detach

We now propose a simple fix to the above problem. Our goal is to manipulate the gradient components such that the components through the cell state path ($\mathbf{A}_t$) do not get suppressed when the components through the remaining paths ($\mathbf{B}_t$) are very large (described in the section 2.2). Thus it would be helpful to multiply $\mathbf{B}_t$ by a positive number less than 1

to dampen its magnitude. In Algorithm 1 we propose a simple trick that achieves this goal. A diagrammatic form of algorithm 1 is shown in Figure 4. In simple words, our algorithm essentially blocks gradients from flowing through each of the $\mathbf{h}_t$ states independently with a probability $1-p$, where $p \in [0,1]$ is a tunable hyper-parameter. Note the subtle detail in Algorithm 1 (line 9) that the loss ℓ_t at any time step t is a function of $\mathbf{h}_t$ which is not detached.

Algorithm 1 Forward Pass of h-detach Algorithm

1: **INPUT:** $\{\mathbf{x}_t\}_{t=1}^{T}$, $\mathbf{h}_0$, $\mathbf{c}_0$, p
2: $\ell = 0$
3: **for** $1 \leq t \leq T$ **do**
4: **if** bernoulli(p)==1 **then**
5: $\tilde{\mathbf{h}}_{t-1} \leftarrow$ stop-gradient($\mathbf{h}_{t-1}$)
6: **else**
7: $\tilde{\mathbf{h}}_{t-1} \leftarrow \mathbf{h}_{t-1}$
8: $\mathbf{h}_t, \mathbf{c}_t \leftarrow \text{LSTM}(\mathbf{x}_t, \tilde{\mathbf{h}}_{t-1}, \mathbf{c}_{t-1})$ (Eq. 2.1- 2.6)
9: $\ell_t \leftarrow \text{loss}(\phi(\mathbf{h}_t))$
10: $\ell \leftarrow \ell + \ell_t$
11: **return** ℓ

We now show that the gradient of the loss function resulting from the LSTM forward pass shown in algorithm 1 has the property that the gradient components arising from $\mathbf{B}_t$ get dampened.

Definition 1. *Let* $\mathbf{z}_t = [\frac{d\mathbf{c}_t}{dw}^T; \frac{d\mathbf{h}_t}{dw}^T; \mathbf{1}_n^T]^T$ *and* $\tilde{\mathbf{z}}_t$ *be the analogue of* $\mathbf{z}_t$ *when applying* h*-detach with probability* $1-p$ *during back-propagation. Then,*

$$\tilde{\mathbf{z}}_t = (\mathbf{A}_t + \xi_t \mathbf{B}_t)(\mathbf{A}_{t-1} + \xi_{t-1}\mathbf{B}_{t-1}) \dots (\mathbf{A}_2 + \xi_2 \mathbf{B}_2)\tilde{\mathbf{z}}_1$$

where $\xi_t, \xi_{t-1}, \dots, \xi_2$ *are i.i.d. Bernoulli random variables with probability* p *of being 1, and* w, $\mathbf{A}_t$ *and* $\mathbf{B}_t$ *and are same as defined in theorem 2.*

The above theorem shows that by stochastically blocking gradients from flowing through the $\mathbf{h}_t$ states of an LSTM with probability $1-p$, we stochastically drop the $\mathbf{B}_t$ term in the gradient components. The corollary below shows that in expectation, this results in dampening the $\mathbf{B}_t$ term compared to the original LSTM gradient.

Corollary 1. $\mathbb{E}_{\xi_2,\dots,\xi_t}[\tilde{\mathbf{z}}_t] = (\mathbf{A}_t + p\mathbf{B}_t)(\mathbf{A}_{t-1} + p\mathbf{B}_{t-1}) \dots (\mathbf{A}_2 + p\mathbf{B}_2)\tilde{\mathbf{z}}_1$

Finally, we note that when training LSTMs with h-detach, we reduce the amount of computation needed. This is simply because by stochastically blocking the gradient from flowing through the h_t hidden states of LSTM, less computation needs to be done during back-propagation through time (BPTT).

3. Experiments

3.1. Copying Task

This task requires the recurrent network to memorize the network inputs provided at the first few time steps and output them in the same order after a large time delay. Thus the only way to solve this task is for the network to capture the long term dependency between inputs and targets which requires gradient components carrying this information to flow through many time steps.

We follow the copying task setup identical to [**2**] (described in appendix). Using their data generation process, we sample 100,000 training input-target sequence pairs and 5,000 validation pairs. We use cross-entropy as our loss to train an LSTM with hidden state size 128 for a maximum of 500-600 epochs. We use the ADAM optimizer with batch-size 100, learning rate 0.001 and clip the gradient norms to 1.

Figure 5 shows the validation accuracy plots for copying task training for $T = 100$ (top row) and $T = 300$ (bottom row) without h-detach (left), and with h-detach (middle and right). Each plot contains runs from the same algorithm with multiple seeds to show a healthy sample of variations using these algorithms. For $T = 100$ time delay, we see both vanilla LSTM and LSTM with h-detach converge to 100% accuracy. For time delay 100 and the training setting used, vanilla LSTM is known to converge to optimal validation performance (for instance, see [**2**]). Nonetheless, we note that h-detach converges faster in this setting. A more interesting case is when time decay is set to 300 because it requires capturing longer term dependencies. In this case, we find that LSTM training without h-detach achieves a validation accuracy of $\sim 82\%$ at best while a number of other seeds converge to much worse performance. On the other hand, we find that using h-detach with detach probabilities 0.25 and 0.5 achieves the best performance of 100% and converging quickly while being reasonably robust to the choice of seed.

3.2. Transfer copying task

Having shown the benefit of h-detach in terms of training dynamics, we now extend the challenge of the copying task by evaluating how well an LSTM trained on data with a certain time delay generalizes when a larger time delay is used during inference. This task is referred as the transfer copying task [**28**]. Specifically, we train the LSTM architecture on copying task with delay $T = 100$ without h-detach and with h-detach with probability 0.25 and 0.5. We then evaluate the accuracy of the trained model for each setting for various values of $T > 100$. The results are shown in table 1. We find that the function learned by LSTM when trained with h-detach generalize significantly better on longer time delays during inference compared with the LSTM trained without h-detach.

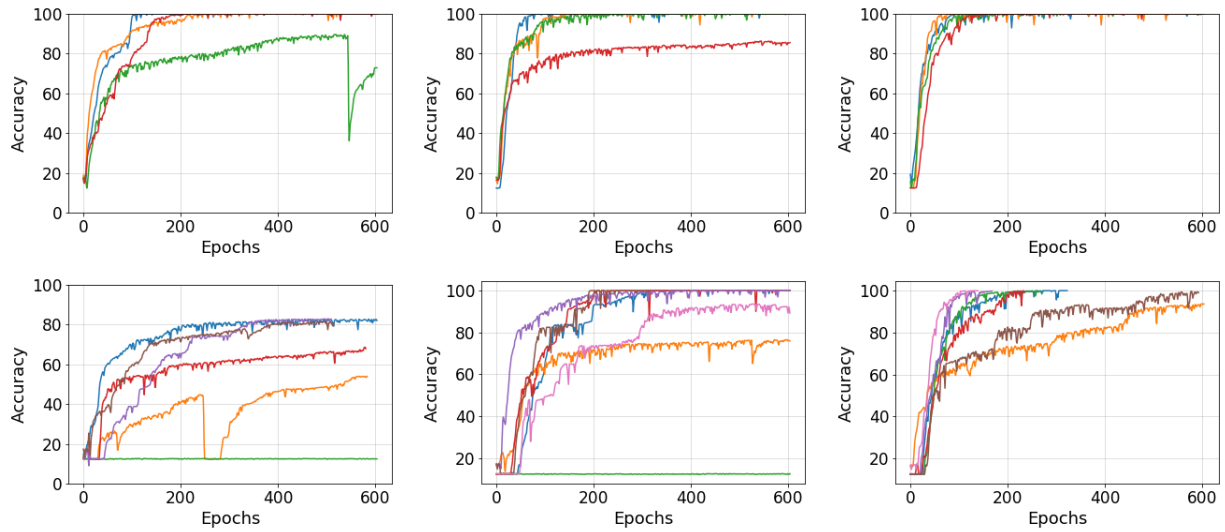

Fig. 5. Validation accuracy curves during training on copying task using vanilla LSTM (left) and LSTM with h-detach with probability 0.25 (middle) and 0.5 (right). **Top row** is delay $T = 100$ and **bottom row** is delay $T = 300$. Each plot contains multiple runs with different seeds. We see that for $T = 100$, even the baseline LSTM is able to reach $\sim 100\%$ accuracy for most seeds and the only difference we see between vanilla LSTM and LSTM with h-detach is in terms of convergence. $T = 300$ is a more interesting case because it involves longer term dependencies. In this case we find that h-detach leads to faster convergence and achieves $\sim 100\%$ validation accuracy while being more robust to the choice of seed.

T	Vanilla LSTM	h-detach 0.5	h-detach 0.25
200	64.85	74.79	**90.72**
400	48.17	54.91	**77.76**
500	43.03	52.43	**74.68**
1000	28.82	43.54	**63.19**
2000	19.48	34.34	**51.83**
5000	14.58	24.55	**42.35**

Table 1. Accuracy on transfer copying task. We find that the generalization of LSTMs trained with h-detach is significantly better compared with vanilla LSTM training when tested on time delays longer that what the model is trained on ($T = 100$).

3.3. Sequential MNIST

This task is a sequential version of the MNIST classification task [**40**]. In this task, an image is fed into the LSTM one pixel per time step and the goal is to predict the label after the last pixel is fed. We consider two versions of the task: one is which the pixels are read in order (from left to right and top to bottom), and one where all the pixels are permuted in a random but fixed order. We call the second version the permuted MNIST task or pMNIST in short. The setup used for this experiment is as follows. We use 50000

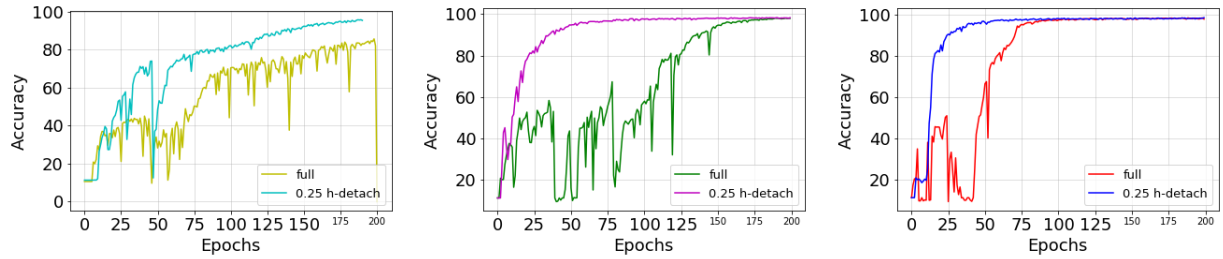

Fig. 6. Validation accuracy curves of LSTM training on pixel by pixel MNIST. Each plot shows LSTM training with and without h-detach for different values of learning rate 0.0001(left), 0.005(middle), 0.001(right). We find that h-detach is both more robust to different learing rates and converges faster compared to vanilla LSTM training. Refer to the Fig. 9 in appendix for validation curves on multiple seeds.

Method	MNIST	pMNIST
Vanilla LSTM	98.5	91.1
SAB [**35**]	-	94.2
iRNN [**39**]	97.0	82.0
uRNN [**2**]	95.1	91.4
Zoneout [**38**]	-	**93.1**
IndRNN [**41**]	**99**	**96**
h-detach (ours)	98.5	92.3

Table 2. A comparison of test accuracy on pixel by pixel MNIST and permuted MNIST (pMNIST) with existing methods.

images for training, 10000 for validation and 10000 for testing. We use the ADAM optimizer with different learning rates– 0.001,0.0005 and 0.0001, and a fixed batch size of 100. We train for 200 epochs and pick our final model based on the best validation score. We use an LSTM with 100 hidden units. For h-detach, we do a hyper-parameter search on the detach probability in $\{0.1, 0.25, 0.4, 0.5\}$. For both pixel by pixel MNIST and pMNIST, we found the detach hyper-parameter of 0.25 to perform best on the validation set for both MNIST and pMNIST.

On the sequential MNIST task, both vanilla LSTM and training with h-detach give an accuracy of 98.5%. Here, we note that the convergence of our method is much faster and is more robust to the different learning rates of the ADAM optimizer as seen in Figure 6. Refer to appendix (figure 9) for experiments with multiple seeds that shows the robustness of our method to initialization.

In the pMNIST task, we find that training LSTM with h-detach gives a test accuracy of 92.3% which is an improvement over the regular LSTM training which reaches an accuracy of 91.1%. A detailed comparison of test performance with existing algorithms is shown in table 2.

3.4. Image Captioning

We now evaluate h-detach on an image captioning task which involves using an RNN for generating captions for images. We use the Microsoft COCO dataset [**42**] which contains 82,783 training images and 40,504 validation images. Since this dataset does not have a standard split for training, validation and test, we follow the setting in [**34**] which suggests a split of 80,000 training images and 5,000 images each for validation and test set.

We use two models to test our approach– the Show&Tell encoder-decoder model [**67**] which does not employ any attention mechanism, and the 'Show, Attend and Tell' model [**71**], which uses soft attention. For feature extraction, we use the 2048-dimensional last layer feature vector of a residual network (Resnet [**24**]) with 152 layers which was pre-trained on ImageNet for image classification. We use an LSTM with 512 hidden units for caption generation. We train both the Resnet and LSTM models using the ADAM optimizer [**36**] with a learning rate of 10^{-4} and leave the rest of the hyper-parameters as suggested in their paper. We also perform a small hyperparameter search where we find the optimial value of the h-detach parameter. We considered values in the set $\{0.1,0.25,0.4,0.5\}$ and pick the optimal value based on the best validation score. Similar to [**61**], we early stop based on the validation CIDEr scores and report BLEU-1 to BLEU-4, CIDEr, and Meteor scores. The BLEU-k score is an metric for evaluating the quality of a sentence, it uses a modified form of precision using k-grams from the sentence when compared to the reference sentence. CIDEr [**66**] and METEOR [**7**] are other well known metrics for measuring the quality of the sentences produced.

The results are presented in table 3. Training the LSTM with h-detach outperforms the baseline LSTM by a good margin for all the metrics and produces the best BLEU-1 to BLEU-3 scores among all the compared methods. Even for the other metrics, except for the results reported by [**45**], we beat all the other methods reported. We emphasize that compared to all the other reported methods, h-detach is extremely simple to implement and does not add any computational overhead (in fact reduces computation).

3.5. Ablation Studies

In this section, we first study the effect of removing gradient clipping in the LSTM training and compare how the training of vanilla LSTM and our method get affected. Getting rid of gradient clipping would be insightful because it would confirm our claim that stochastically blocking gradients through the hidden states h_t of the LSTM prevent the growth of gradient magnitude. We train both models on pixel by pixel MNIST using ADAM without any gradient clipping. The validation accuracy curves are reported in figure 7 for two different learning rates. We notice that removing gradient clipping causes the Vanilla LSTM training to become extremely unstable, there are sudden drops in the validation accuracy at

Table 3. Test performance on image captioning task on MS COCO dataset using metrics BLEU 1 to 4, METEOR, and CIDEr (higher values are better for all metrics). We re-implement both Show&Tell [**67**] and Soft Attention [**71**] and train the LSTM in these models with and without h-detach.

Models	B-1	B-2	B-3	B-4	METEOR	CIDEr
DeepVS [**34**]	62.5	45.0	32.1	23.0	19.5	66.0
ATT-FCN [**74**]	70.9	53.7	40.2	30.4	24.3	—
Show & Tell [**67**]	—	—	—	27.7	23.7	85.5
Soft Attention [**71**]	70.7	49.2	34.4	24.3	23.9	—
Hard Attention [**71**]	71.8	50.4	35.7	25.0	23.0	—
MSM [**73**]	73.0	56.5	42.9	32.5	25.1	98.6
Adaptive Attention [**45**]	**74.2**	**58.0**	**43.9**	**33.2**	**26.6**	**108.5**
TwinNet [**61**]						
No attention, Resnet152	72.3	55.2	40.4	29.3	25.1	94.7
Soft Attention, Resnet152	73.8	56.9	42.0	30.6	25.2	97.3
No attention, Resnet152						
Show&Tell (Our impl.)	71.7	54.4	39.7	28.8	24.8	93.0
+ h-detach (0.25)	**72.9**	**55.8**	**41.7**	**31.0**	**25.1**	**98.0**
Attention, Resnet152						
Soft Attention (Our impl.)	73.2	56.3	41.4	30.1	25.3	96.6
+ h-detach (0.4)	**74.7**	**58.1**	**44.0**	**33.1**	**26.0**	**103.3**

several points in the training. h-detach on the other hand seems robust to removing gradient clipping for both the learning rates used. Additional experiments with multiple seeds and learning rates can be found in figure 11 in appendix.

Second, we conduct experiments where we stochastically block gradients from flowing through the cell state $\mathbf{c}_t$ instead of the hidden state $\mathbf{h}_t$ and observe how the LSTM behaves in such a scenario. We refer detaching the cell state as c-detach. The goal of this experiment is to corroborate our hypothesis that the gradients through the cell state path carry information about long term dependencies. Figure 8 shows the effect of c-detach (with probabilities shown) on copying task and pixel by pixel MNIST task. We notice in the copying task for $T = 100$, learning becomes very slow (figure 8) and does not converge even after 500 epochs, whereas when not detaching the cell state, even the Vanilla LSTM converges in around 150 epochs for most cases for T=100 as shown in the experiments in section 3.1. For pixel by pixel MNIST (which involves 784 time steps), there is a much larger detrimental effect on learning as we find that none of the seeds cross 60% accuracy at the end of training (Figure 8 (b)). This experiment corroborates our hypothesis that gradients through the cell state contain important components of the gradient signal as blocking them worsens the performance of these models when compared to Vanilla LSTM.

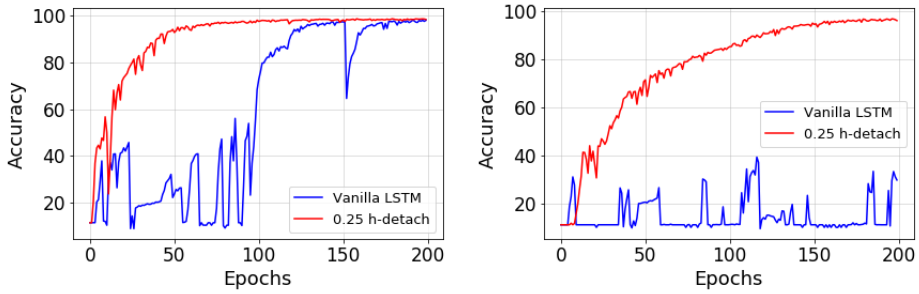

Fig. 7. The effect of removing gradient clipping from vanilla LSTM training vs. LSTM trained with h-detach on pixel by pixel MNIST dataset for two learning rates 0.0005(left) and 0.0001(right). Refer to Fig. 11 in appendix for experiments with multiple seeds.

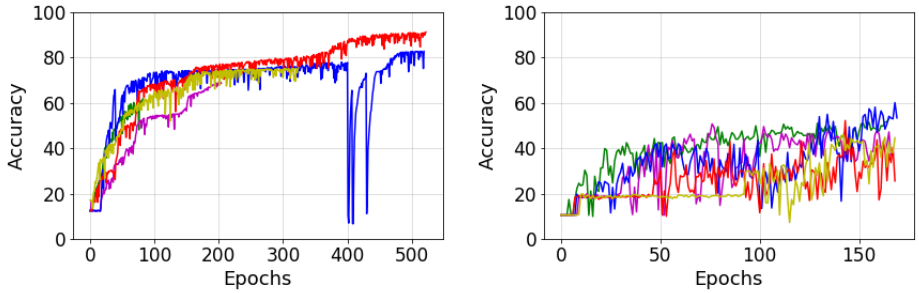

Fig. 8. Validation accuracy curves for copying task T=100 (left) and pixel by pixel MNIST (right) using LSTM such that gradient is stochastically blocked through the cell state (the probability of detaching the cell state in this experiment is mentioned in sub-titles.). Blocking gradients from flowing through the cell state path of LSTM (c-detach) leads to significantly worse performance compared even to vanilla LSTM on tasks that requires long term dependencies.

4. Related Work

Capturing long term dependencies in data using recurrent neural networks has been long known to be a hard problem [**26, 9**]. Therefore, there has been a considerable amount of work on addressing this issue. Prior to the invention of the LSTM architecture [**28**], another class of architectures called NARX (nonlinear autoregressive models with exogenous) recurrent networks [**43**] was popular for tasks involving long term dependencies. More recently gated recurrent unit (GRU) networks [**15**] was proposed that adapts some favorable properties of LSTM while requiring fewer parameters. Other recent recurrent architecture designs that are aimed at preventing EVGP can be found in [**76**], [**33**] and [**41**]. Work has also been done towards better optimization for such tasks [**48, 36**]. Since vanishing and exploding gradient problems [**26, 10**] also hinder this goal, gradient clipping methods have been proposed to alleviate this problem [**63, 53**]. Yet another line of work focuses on making use of unitary transition matrices in order to avoid loss of information as hidden states evolve over time. [**39**] propose to initialize recurrent networks with unitary weights while [**2**] propose a new

network parameterization that ensures that the state transition matrix remains unitary. Extensions of the unitary RNNs have been proposed in [**70**], [**49**] and [**32**]. Very recently, [**35**] propose to learn an attention mechanism over past hidden states and sparsely back-propagate through paths with high attention weights in order to capture long term dependencies. [**64**] propose to add an unsupervised auxiliary loss to the original objective that is designed to encourage the network to capture such dependencies. We point out that our proposal in this paper is orthogonal to a number of the aforementioned papers and may even be applied in conjunction to some of them. Further, our method is specific to LSTM optimization and reduces computation relative to the vanilla LSTM optimization which is in stark contrast to most of the aforementioned approaches which increase the amount of computation needed for training.

5. Discussion and Future Work

In section 3.5 we showed that LSTMs trained with h-detach are stable even without gradient clipping. We caution that while this is true, in general the gradient magnitude depends on the value of detaching probability used in h-detach. Hence for the general case, we do not recommend removing gradient clipping.

When training stacked LSTMs, there are two ways in which h-detach can be used: 1) detaching the hidden state of all LSTMs simultaneously for a given time step t depending on the stochastic variable ξ_t) stochastically detaching the hidden state of each LSTM separately. We leave this for future work.

h-detach stochastically blocks the gradient from flowing through the hidden states of LSTM. In corollary 1, we showed that in expectation, this is equivalent to dampening the gradient components from paths other than the cell state path. We especially chose this strategy because of its ease of implementation in current auto-differentiation libraries. Another approach to dampen these gradient components would be to directly multiply these components with a dampening factor. This feature is currently unavailable in these libraries but may be an interesting direction to look into. A downside of using this strategy though is that it will not reduce the amount of computation similar to h-detach (although it will not increase the amount of computation compared with vanilla LSTM either). Regularizing the recurrent weight matrices to have small norm can also potentially prevent the gradient components from the cell state path from being suppressed but it may also restrict the representational power of the model.

Given the superficial similarity of h-detach with dropout, we outline the difference between the two methods. Dropout randomly masks the hidden units of a network during the forward pass (and can be seen as a variant of the stochastic delta rule [**22**]). Therefore, a common view of dropout is training an ensemble of networks [**68**]. On the other hand, our

method does not mask the hidden units during the forward pass. It instead randomly blocks the gradient component through the h-states of the LSTM only during the backward pass and does not change the output of the network during forward pass. More specifically, our theoretical analysis shows the precise behavior of our method: the effect of h-detach is that it changes the update direction used for descent which prevents the gradients through the cell state path from being suppressed.

We would also like to point out that even though we show improvements on the image captioning task, it does not fit the profile of a task involving long term dependencies that we focus on. We believe the reason why our method leads to improvements on this task is that the gradient components from the cell state path are important for this task and our theoretical analysis shows that h-detach prevents these components from getting suppressed compared with the gradient components from the other paths. On the same note, we also tried our method on language modeling tasks but did not notice any benefit.

6. Conclusion

We proposed a simple stochastic algorithm called h-detach aimed at improving LSTM performance on tasks that involve long term dependencies. We provided a theoretical understanding of the method using a novel analysis of the back-propagation equations of the LSTM architecture. We note that our method reduces the amount of computation needed during training compared to vanilla LSTM training. Finally, we empirically showed that h-detach is robust to initialization, makes the convergence of LSTM faster, and/or improves generalization compared to vanilla LSTM (and other existing methods) on various benchmark datasets.

7. Appendix

7.1. Additional Information

Copying Experiment setup - We define 10 tokens, $\{a_i\}_{i=0}^{9}$. The input to the LSTM is a sequence of length $T+20$ formed using one of the ten tokens at each time step. Input for the first 10 time steps are sampled i.i.d. (uniformly) from $\{a_i\}_{i=0}^{7}$. The next $T-1$ entries are set to a_8, which constitutes a delay. The next single entry is a_9, which represents a delimiter, which should indicate to the algorithm that it is now required to reproduce the initial 10 input tokens as output. The remaining 10 input entries are set to a_8. The target sequence consists of $T+10$ repeated entries of a_8, followed by the first 10 entries of the input sequence in exactly the same order.

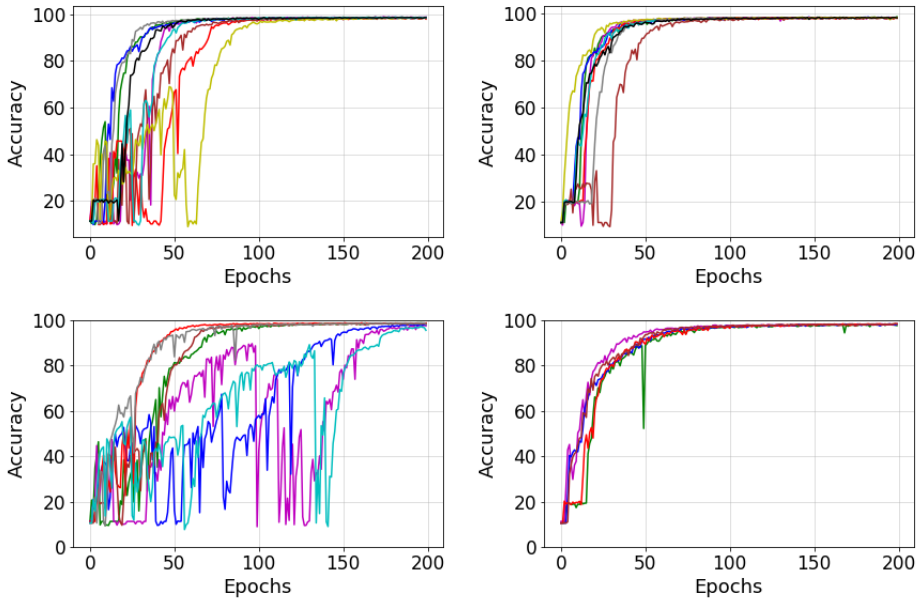

Fig. 9. Validation accuracy curves on pixel by pixel MNIST dataset with vanilla LSTM training and LSTM training with h-detach with various values of learning rate and initialization seeds. Vanilla LSTM is on the left and h-detach 0.25 is on the right. The top row has a learning rate of 0.001 while the bottom row has 0.0005.

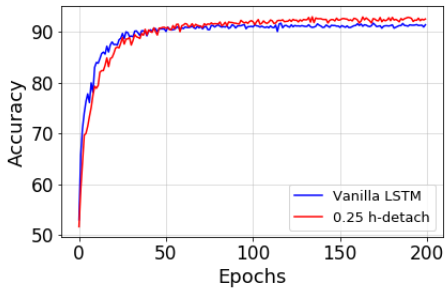

Fig. 10. Validation accuracy curves on pMNIST dataset with vanilla LSTM training and LSTM training with h-detach.

7.2. Derivation of Back-propagation Equation for LSTM

Let us recall the equations from an LSTM

$$\mathbf{o}_t = \sigma\left(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t]^T + \mathbf{b}_o\right)$$
$$\mathbf{i}_t = \sigma\left(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t]^T + \mathbf{b}_i\right)$$
$$\mathbf{g}_t = \tanh\left(\mathbf{W}_g[\mathbf{h}_{t-1}, \mathbf{x}_t]^T + \mathbf{b}_g\right)$$
$$\mathbf{f}_t = \sigma\left(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t]^T + \mathbf{b}_f\right)$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

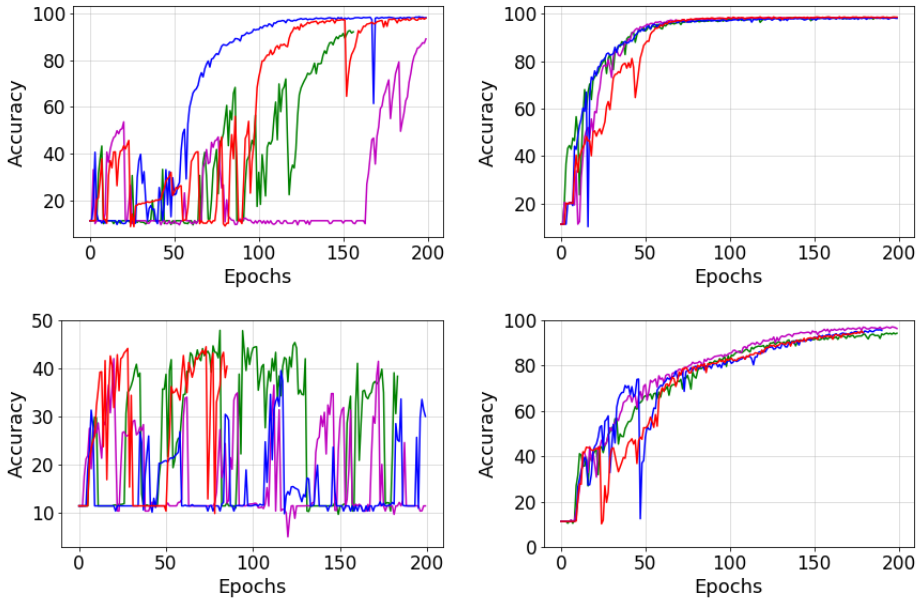

Fig. 11. The effect of removing gradient clipping during optimization. Validation accuracy curves on pixel by pixel MNIST dataset with vanilla LSTM training and LSTM training with h-detach with various values of learning rate and initialization seeds. LSTM training using h-detach is both significantly more stable and robust to initialization when removing gradient clipping compared with vanilla LSTM training. Vanilla LSTM is on the left and h-detach 0.25 is on the right. The top row has a learning rate of 0.001 while the bottom row has 0.0005.

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t$$

Here $\odot$ denotes the element-wise product, also called the Hadamard product. σ denotes the sigmoid activation function. $\mathbf{W}_o = [\mathbf{W}_{oh}; \mathbf{W}_{ox}]$. $\mathbf{W}_i = [\mathbf{W}_{ih}; \mathbf{W}_{ix}]$. $\mathbf{W}_g = [\mathbf{W}_{gh}; \mathbf{W}_{gx}]$. $\mathbf{W}_f = [\mathbf{W}_{fh}; \mathbf{W}_{fx}]$.

Notation 1.

$$\mathbf{\Delta}_t^c = diag[\mathbf{o}_t \odot (1 - \tanh^2(\mathbf{c}_t))]$$

$$\mathbf{\Delta}_t^o = diag[\mathbf{o}_t \odot (1 - \mathbf{o}_t) \odot \tanh(\mathbf{c}_t)]$$

$$\mathbf{\Delta}_t^f = diag[\mathbf{f}_t \odot (1 - \mathbf{f}_t) \odot \mathbf{c}_{t-1}]$$

$$\mathbf{\Delta}_t^i = diag[\mathbf{i}_t \odot (1 - \mathbf{i}_t) \odot \mathbf{g}_t]$$

$$\mathbf{\Delta}_t^g = diag[(1 - \mathbf{g}_t^2) \odot \mathbf{i}_t]$$

For any $\star \in \{f,g,o,i\}$, *define* $\mathbf{E}^{\star}(w)$ *to be a matrix of size dim(*$\mathbf{h}_t$*)* $\times$ *dim(*$[\mathbf{h}_t;\mathbf{x}_t]$*). We set all the elements of this matrix to* 0*s if if* w *is not an element of* $\mathbf{W}_{\star}$. *Further, if* $w = (\mathbf{W}_{\star})_{kl}$, *then* $(\mathbf{E}^{\star}(w))_{kl} = 1$ *and* $(\mathbf{E}^{\star}(w))_{k'l'} = 0$ *for all* $(k',l') \neq (k,l)$.

$$\psi_t = \mathbf{\Delta}_t^f \mathbf{W}_{fh} + \mathbf{\Delta}_t^g \mathbf{W}_{gh} + \mathbf{\Delta}_t^i \mathbf{W}_{ih}$$

$$\tilde{\psi}_t = \mathbf{\Delta}_t^o \mathbf{W}_{oh} + \mathbf{\Delta}_t^c \psi_t$$

$$\mathbf{k}_t = \left(\mathbf{\Delta}_t^f \mathbf{E}^f(w) + \mathbf{\Delta}_t^g \mathbf{E}^g(w) + \mathbf{\Delta}_t^i \mathbf{E}^i(w)\right) \cdot [\mathbf{h}_{t-1},\mathbf{x}_t]^T$$

$$\tilde{\mathbf{k}}_t = \mathbf{\Delta}_t^o \mathbf{E}^o(w) \cdot [\mathbf{h}_{t-1},\mathbf{x}_t]^T + \mathbf{\Delta}_t^c \mathbf{k}_t$$

$$\mathbf{F}_t = diag(\mathbf{f}_t)$$
$$\tilde{\mathbf{F}}_t = \mathbf{\Delta}_t^c diag(\mathbf{f}_t)$$

Lemma 1. *Let us assume* w *is an entry of the matrix* $\mathbf{W}_f, \mathbf{W}_i, \mathbf{W}_g$ *or* $\mathbf{W}_o$, *then*

$$\frac{d\mathbf{f}_t}{dw} = diag[\mathbf{f}_t \odot (1 - \mathbf{f}_t)] \cdot \left(\mathbf{W}_{fh} \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \mathbf{E}^f(w) \cdot [\mathbf{h}_{t-1},\mathbf{x}_t]^T\right)$$

$$\frac{d\mathbf{o}_t}{dw} = diag[\mathbf{o}_t \odot (1 - \mathbf{o}_t)] \cdot \left(\mathbf{W}_{oh} \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \mathbf{E}^o(w) \cdot [\mathbf{h}_{t-1},\mathbf{x}_t]^T\right)$$

$$\frac{d\mathbf{i}_t}{dw} = diag[\mathbf{i}_t \odot (1 - \mathbf{i}_t)] \cdot \left(\mathbf{W}_{ih} \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \mathbf{E}^i(w) \cdot [\mathbf{h}_{t-1},\mathbf{x}_t]^T\right)$$

$$\frac{d\mathbf{g}_t}{dw} = diag[(1 - \mathbf{g}_t^2)] \cdot \left(\mathbf{W}_{gh} \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \mathbf{E}^g(w) \cdot [\mathbf{h}_{t-1},\mathbf{x}_t]^T\right)$$

Proof *By chain rule of total differentiation,*

$$\frac{d\mathbf{f}_t}{dw} = \frac{\partial \mathbf{f}_t}{\partial w} + \frac{\partial \mathbf{f}_t}{\partial \mathbf{h}_{t-1}} \frac{d\mathbf{h}_{t-1}}{dw}$$

We note that,

$$\frac{\partial \mathbf{f}_t}{\partial w} = diag[\mathbf{f}_t \odot (1 - \mathbf{f}_t)] \cdot \mathbf{E}^f(w) \cdot [\mathbf{h}_{t-1},\mathbf{x}_t]^T$$

and,

$$\frac{\partial \mathbf{f}_t}{\partial \mathbf{h}_{t-1}} = diag[\mathbf{f}_t \odot (1 - \mathbf{f}_t)] \cdot \mathbf{W}_{fh} \cdot \frac{d\mathbf{h}_{t-1}}{dw}$$

which proves the claim for $\frac{d\mathbf{f}_t}{dw}$. *The derivation for* $\frac{d\mathbf{o}_t}{dw}, \frac{d\mathbf{i}_t}{dw}, \frac{d\mathbf{g}_t}{dw}$ *are similar.*

Now let us establish recursive formulas for $\frac{d\mathbf{h}_t}{dw}$ and $\frac{d\mathbf{c}_t}{dw}$, using the above formulas.

Corollary 2. *Considering the above notations, we have*

$$\frac{d\mathbf{h}_t}{dw} = \mathbf{\Delta}_t^o \mathbf{W}_{oh} \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \mathbf{\Delta}_t^c \cdot \frac{d\mathbf{c}_t}{dw} + \mathbf{\Delta}_t^o \mathbf{E}^o(w) \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]^T$$

Proof *Recall that* $\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$*, and thus*

$$\frac{d\mathbf{h}_t}{dw} = \frac{d\mathbf{o}_t}{dw} \odot \tanh(\mathbf{c}_t) + \mathbf{o}_t \odot (1 - \tanh^2(\mathbf{c}_t)) \odot \frac{d\mathbf{c}_t}{dw}$$

Using the previous Lemma as well as the above notation, we get

$$\frac{d\mathbf{h}_t}{dw} = diag[\mathbf{o}_t \odot (1-\mathbf{o}_t)] \cdot \left(\mathbf{W}_{oh} \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \mathbf{E}^o(w) \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]^T \right) \odot \tanh(\mathbf{c}_t) + \mathbf{o}_t \odot (1 - \tanh^2(\mathbf{c}_t)) \odot \frac{d\mathbf{c}_t}{dw}$$

$$= \mathbf{\Delta}_t^o \mathbf{W}_{oh} \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \mathbf{\Delta}_t^o \mathbf{E}^o(w) \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]^T + \mathbf{o}_t \odot (1 - \tanh^2(\mathbf{c}_t)) \odot \frac{d\mathbf{c}_t}{dw}$$

$$= \mathbf{\Delta}_t^c \frac{d\mathbf{c}_t}{dw} + \mathbf{\Delta}_t^o \mathbf{W}_{oh} \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \mathbf{\Delta}_t^o \mathbf{E}^o(w) \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]^T$$

Corollary 3. *Considering the above notations, we have*

$$\frac{d\mathbf{c}_t}{dw} = \mathbf{F}_t \frac{d\mathbf{c}_{t-1}}{dw} + \psi_t \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \mathbf{k}_t$$

Proof *Recall that* $\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t$*, and thus*

$$\frac{d\mathbf{c}_t}{dw} = \frac{d\mathbf{f}_t}{dw} \odot \mathbf{c}_{t-1} + \mathbf{f}_t \odot \frac{d\mathbf{c}_{t-1}}{dw} + \frac{d\mathbf{g}_t}{dw} \odot \mathbf{i}_t + \mathbf{g}_t \odot \frac{d\mathbf{i}_t}{dw}$$

Using the previous Lemma as well as the above notation, we get

$$\begin{aligned}
\frac{d\mathbf{c}_t}{dw} = & \, diag[\mathbf{f}_t \odot (1 - \mathbf{f}_t)] \cdot \left(\mathbf{W}_{fh} \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \mathbf{E}^f(w) \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]^T \right) \odot \mathbf{c}_{t-1} + \mathbf{f}_t \odot \frac{d\mathbf{c}_{t-1}}{dw} \\
& + diag[(1 - \mathbf{g}_t^2)] \cdot \left(\mathbf{W}_{gh} \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \mathbf{E}^g(w) \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]^T \right) \odot \mathbf{i}_t \\
& + diag[\mathbf{i}_t \odot (1 - \mathbf{i}_t)] \cdot \left(\mathbf{W}_{ih} \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \mathbf{E}^i(w) \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]^T \right) \odot \mathbf{g}_t \\
= & \, \mathbf{\Delta}_t^f \mathbf{W}_{fh} \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \mathbf{\Delta}_t^f \mathbf{E}^f(w) \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]^T + \mathbf{f}_t \odot \frac{d\mathbf{c}_{t-1}}{dw} \\
& + \mathbf{\Delta}_t^g \mathbf{W}_{gh} \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \mathbf{\Delta}_t^g \mathbf{E}^g(w) \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]^T \\
& + \mathbf{\Delta}_t^i \mathbf{W}_{ih} \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \mathbf{\Delta}_t^i \mathbf{E}^i(w) \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]^T \\
= & \, \mathbf{F}_t \frac{d\mathbf{c}_{t-1}}{dw} + \left(\mathbf{\Delta}_t^f \mathbf{W}_{fh} + \mathbf{\Delta}_t^g \mathbf{W}_{gh} + \mathbf{\Delta}_t^i \mathbf{W}_{ih} \right) \cdot \frac{d\mathbf{h}_{t-1}}{dw} \\
& + \left(\mathbf{\Delta}_t^f \mathbf{E}^g(w) + \mathbf{\Delta}_t^g \mathbf{E}^g(w) + \mathbf{\Delta}_t^i \mathbf{E}^g(w) \right) \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]^T
\end{aligned}$$

$$= \mathbf{F}_t \frac{d\mathbf{c}_{t-1}}{dw} + \psi_t \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \mathbf{k}_t$$

Let us now combine corollary 1 and 2 to get a recursive expression of $\frac{dh_t}{dw}$ in terms of $\frac{d\mathbf{h}_{t-1}}{dw}$ and $\frac{d\mathbf{c}_{t-1}}{dw}$

Corollary 4. *Considering the above notations, we have*

$$\frac{d\mathbf{h}_t}{dw} = \tilde{\mathbf{F}}_t \frac{d\mathbf{c}_{t-1}}{dw} + \tilde{\psi}_t \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \tilde{\mathbf{k}}_t$$

Proof *From Corollary 1, we know that*

$$\frac{d\mathbf{h}_t}{dw} = \mathbf{\Delta}_t^o \mathbf{W}_{oh} \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \mathbf{\Delta}_t^c \cdot \frac{d\mathbf{c}_t}{dw} + \mathbf{\Delta}_t^o \mathbf{E}^o(w) \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]^T$$

Using Corollary 2, we get

$$\frac{d\mathbf{h}_t}{dw} = \mathbf{\Delta}_t^o \mathbf{W}_{oh} \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \mathbf{\Delta}_t^c \cdot \left(\mathbf{F}_t \frac{d\mathbf{c}_{t-1}}{dw} + \psi_t \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \mathbf{k}_t \right) + \mathbf{\Delta}_t^o \mathbf{E}^o(w) \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]^T$$

$$= \mathbf{\Delta}_t^c \cdot \mathbf{F}_t \frac{d\mathbf{c}_{t-1}}{dw} + (\mathbf{\Delta}_t^o \mathbf{W}_{oh} + \psi_t) \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \left(\mathbf{k}_t + \mathbf{\Delta}_t^o \mathbf{E}^o(w) \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]^T \right)$$

$$= \tilde{\mathbf{F}}_t \frac{d\mathbf{c}_{t-1}}{dw} + \tilde{\psi}_t \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \tilde{\mathbf{k}}_t$$

Theorem 2. *Fix w to be an element of the matrix $\mathbf{W}_{gh}, \mathbf{W}_{fh}, \mathbf{W}_{ih}$ or $\mathbf{W}_{oh}$. Define,*

$$\mathbf{A}_t = \begin{bmatrix} \mathbf{F}_t & \mathbf{0}_n & diag(\mathbf{k}_t) \\ \tilde{\mathbf{F}}_t & \mathbf{0}_n & diag(\tilde{\mathbf{k}}_t) \\ \mathbf{0}_n & \mathbf{0}_n & \mathbf{Id}_n \end{bmatrix} \quad \mathbf{B}_t = \begin{bmatrix} \mathbf{0}_n & \psi_t & \mathbf{0}_n \\ \mathbf{0}_n & \tilde{\psi}_t & \mathbf{0}_n \\ \mathbf{0}_n & \mathbf{0}_n & \mathbf{0}_n \end{bmatrix} \quad \mathbf{z}_t = \begin{bmatrix} \frac{d\mathbf{c}_t}{dw} \\ \frac{d\mathbf{h}_t}{dw} \\ \mathbf{1}_n \end{bmatrix} \tag{7.1}$$

Then,

$$\mathbf{z}_t = (\mathbf{A}_t + \mathbf{B}_t)\mathbf{z}_{t-1}$$

In other words,

$$\mathbf{z}_t = (\mathbf{A}_t + \mathbf{B}_t)(\mathbf{A}_{t-1} + \mathbf{B}_{t-1}) \dots (\mathbf{A}_2 + \mathbf{B}_2)\mathbf{z}_1$$

where all the symbols used to define $\mathbf{A}_t$ and $\mathbf{B}_t$ are defined in notation 1.

Proof *By Corollary 2, we get*

$$\frac{d\mathbf{c}_t}{dw} = \mathbf{F}_t \frac{d\mathbf{c}_{t-1}}{dw} + \psi_t \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \mathbf{k}_t$$

$$= \mathbf{F}_t \frac{d\mathbf{c}_{t-1}}{dw} + \psi_t \cdot \frac{d\mathbf{h}_{t-1}}{dw} + diag(\mathbf{k}_t)\mathbf{1}_n$$

$$= \begin{bmatrix} \mathbf{F}_t & \psi_t & diag(\mathbf{k}_t) \end{bmatrix} \cdot \mathbf{z}_{t-1}$$

Similarly by Corollary 3, we get

$$\frac{d\mathbf{h}_t}{dw} = \tilde{\mathbf{F}}_t \frac{d\mathbf{c}_{t-1}}{dw} + \tilde{\psi}_t \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \tilde{\mathbf{k}}_t$$

$$= \tilde{\mathbf{F}}_t \frac{d\mathbf{c}_{t-1}}{dw} + \tilde{\psi}_t \cdot \frac{d\mathbf{h}_{t-1}}{dw} + diag(\tilde{\mathbf{k}}_\mathbf{t})\mathbf{1}_n$$

$$= \begin{bmatrix} \tilde{\mathbf{F}}_t & \tilde{\psi}_t & diag(\tilde{\mathbf{k}}_t) \end{bmatrix} \cdot \mathbf{z}_{t-1}$$

Thus we have

$$\mathbf{z}_t = \begin{bmatrix} \mathbf{F}_t & \psi_t & diag(\mathbf{k}_t) \\ \tilde{\mathbf{F}}_t & \tilde{\psi}_t & diag(\tilde{\mathbf{k}}_\mathbf{t}) \\ \mathbf{0}_n & \mathbf{0}_n & \mathbf{Id}_n \end{bmatrix} \cdot \mathbf{z}_{t-1} = (\mathbf{A}_t + \mathbf{B}_t) \cdot \mathbf{z}_{t-1} \tag{7.2}$$

Applying this formula recursively proves the claim.

Note: Since $\mathbf{A}_t$ has $\mathbf{0}_n$'s in the second column of the block matrix representation, it ignores the contribution of $\mathbf{z}_t$ coming from $\mathbf{h}_{t-1}$, whereas $\mathbf{B}_t$ (having non-zero block matrices only in the second column of the block matrix representation) only takes into account the contribution coming from $\mathbf{h}_{t-1}$. Hence $\mathbf{A}_t$ captures the contribution of the gradient coming from the cell state $\mathbf{c}_{t-1}$.

7.3. Derivation of Back-propagation Equation for LSTM with *h*-detach

Theorem 3. *Let* $\mathbf{z}_t = [\frac{d\mathbf{c}_t}{dw}^T; \frac{d\mathbf{h}_t}{dw}^T; \mathbf{1}_n^T]^T$ *and* $\tilde{\mathbf{z}}_t$ *be the analogue of* $\mathbf{z}_t$ *when applying h-detach with probability p during back-propagation. Then,*

$$\tilde{\mathbf{z}}_t = (\mathbf{A}_t + \xi_t \mathbf{B}_t)(\mathbf{A}_{t-1} + \xi_{t-1}\mathbf{B}_{t-1}) \dots (\mathbf{A}_2 + \xi_2 \mathbf{B}_2)\tilde{\mathbf{z}}_1$$

where $\xi_t, \xi_{t-1}, \dots, \xi_2$ *are i.i.d. Bernoulli random variables with probability p of being 1,* $\mathbf{A}_t$ *and* $\mathbf{B}_t$ *and are same as defined in theorem 2.*

Proof *Replacing* $\frac{\partial}{\partial \mathbf{h}_{t-1}}$ *by* $\xi_t \frac{\partial}{\partial \mathbf{h}_{t-1}}$ *in lemma 1 and therefore in Corollaries 2 and 3, we get the following analogous equations*

$$\frac{d\mathbf{c}_t}{dw} = \mathbf{F}_t \frac{d\mathbf{c}_{t-1}}{dw} + \xi_t \psi_t \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \mathbf{k}_t$$

and

$$\frac{d\mathbf{h}_t}{dw} = \tilde{\mathbf{F}}_t \frac{d\mathbf{c}_{t-1}}{dw} + \xi_t \tilde{\psi}_t \cdot \frac{d\mathbf{h}_{t-1}}{dw} + \tilde{\mathbf{k}}_t$$

Similarly as in the proof of previous theorem, we can rewrite

$$\frac{d\mathbf{c}_t}{dw} = \begin{bmatrix} \mathbf{F}_t & \xi_t \psi_t & diag(\mathbf{k}_t) \end{bmatrix} \cdot \tilde{\mathbf{z}}_{t-1}$$

and

$$\frac{d\mathbf{h}_t}{dw} = \begin{bmatrix} \tilde{\mathbf{F}}_t & \xi_t \tilde{\psi}_t & diag(\tilde{\mathbf{k}}_t) \end{bmatrix} \cdot \tilde{\mathbf{z}}_{t-1}$$

Thus

$$\tilde{\mathbf{z}}_t = \begin{bmatrix} \mathbf{F}_t & \xi_t \psi_t & diag(\mathbf{k}_t) \\ \tilde{\mathbf{F}}_t & \xi_t \tilde{\psi}_t & diag(\tilde{\mathbf{k}}_t) \\ \mathbf{0}_n & \mathbf{0}_n & \mathbf{Id}_n \end{bmatrix} \cdot \tilde{\mathbf{z}}_{t-1} = \left(\begin{bmatrix} \mathbf{F}_t & \mathbf{0}_n & diag(\mathbf{k}_t) \\ \tilde{\mathbf{F}}_t & \mathbf{0}_n & diag(\tilde{\mathbf{k}}_t) \\ \mathbf{0}_n & \mathbf{0}_n & \mathbf{Id}_n \end{bmatrix} + \xi_t \begin{bmatrix} \mathbf{0}_n & \psi_t & \mathbf{0}_n \\ \mathbf{0}_n & \tilde{\psi}_t & \mathbf{0}_n \\ \mathbf{0}_n & \mathbf{0}_n & \mathbf{0}_n \end{bmatrix} \right) \cdot \tilde{\mathbf{z}}_{t-1}$$

$$= (\mathbf{A}_t + \xi_t \mathbf{B}_t) \cdot \tilde{\mathbf{z}}_{t-1}$$

Iterating this formula gives,

$$\tilde{\mathbf{z}}_t = (\mathbf{A}_t + \xi_t \mathbf{B}_t)(\mathbf{A}_{t-1} + \xi_{t-1} \mathbf{B}_{t-1}) \dots (\mathbf{A}_3 + \xi_3 \mathbf{B}_3) \tilde{\mathbf{z}}_2$$

Corollary 5.

$$\mathbb{E}[\tilde{\mathbf{z}}_t] = (\mathbf{A}_t + p\mathbf{B}_t)(\mathbf{A}_{t-1} + p\mathbf{B}_{t-1}) \dots (\mathbf{A}_3 + p\mathbf{B}_3) \tilde{\mathbf{z}}_2$$

It suffices to take the expectation both sides, and use independence of ξ_t's.

Second Article.

Untangling trade-offs between recurrence and self-attention in neural networks

by

Bhargav Kanuparthi[*,1], Giancarlo Kerg[*,1], Anirudh Goyal[1],
Kyle Goyette[1], Yoshua Bengio[1], and Guillaume Lajoie[1]

([1]) Université de Montréal, Département d'informatique et de recherche opérationelle, Montreal, Canada and Mila - Quebec AI Institute

This article was submitted and is under review as a conference paper at **NeurIPS 2020**.

The main contributions of Bhargav Kanuparthi for this article are presented.

- Proposed the initial idea for the *Relevancy Screening Mechanism*.
- Implementation of the RelRNN / RelLSTM algorithms, experiments and hyperparameter searches to find the best results on standard benchmark tasks.
- Involved in discussions and implementation of analysis section to get a deeper understanding of the algorithm.
- Involved in discussions and helped with proofs for main theorems.
- Writing of the *Relevancy Screening Mechanism*, Experimental Results and Analysis sections of the submitted paper.

Abstract. Attention and self-attention mechanisms, inspired by cognitive processes, are now central to state-of-the-art deep learning on sequential tasks. However, most recent progress hinges on heuristic approaches with limited understanding of attention's role in model optimization and computation, and rely on considerable memory and computational resources that scale poorly. In this work, we present a formal analysis of how self-attention affects gradient propagation in recurrent networks, and prove that it mitigates the problem of vanishing gradients when trying to capture long-term dependencies. Building on these results, we propose a relevancy screening mechanism, inspired by the cognitive process of memory consolidation, that allows for a scalable use of sparse self-attention with recurrence. While providing guarantees to avoid vanishing gradients, we use simple numerical experiments to demonstrate the tradeoffs in performance and computational resources by efficiently balancing attention and recurrence. Based on our results, we propose a concrete direction of research to improve scalability of attentive networks.
Keywords: Exploding and vanishing gradients problem, self-attention, scalablity, gradient propagation

1. Introduction

We live in a world where most of the information takes a sequential form, largely because it is delivered over time. Performing computations on streams of sequential inputs requires extracting relevant temporal dependencies and learning to recognize patterns across several timescales. Humans can effortlessly make associations relating events stored in memory which are far from each other in time and thus, capture long-term dependencies.

Historically, recurrent neural networks (RNNs) have been the deep network architecture of choice for this type of task since, just like neural circuits in the brain, they enable *dynamics* that can be shaped to interact with input streams. However, RNNs (including gated RNNs [**59, 15**]) still struggle with large timescales as their iterative nature leads to unstable information propagation [**8, 52, 59, 27**].This is because most standard RNNs rely on their current state h_t, a vector of fixed dimension, to represent a summary of relevant past information. Indeed, [**8**] showed that without making additional assumptions, storing information in a fixed-size state vector in a stable way necessarily leads to vanishing gradients when back-propagating through time (see also [**27**]).

Several attempts have been made to augment RNN dynamics with external memory to mitigate these issues [**62, 20, 58, 21**], but it is only recently that access to externally stored information has become effective with the introduction of *attention*, and more particularly *soft attention* mechanisms [**5**]. Attention provides a way by which a system can dynamically access past states and inputs across several timescales, bypassing the need of sequential propagation and ignoring irrelevant information (or distractor information). There is substantial empirical evidence that attention, especially *self-attention* ([**65, 35**]), is very helpful to improve learning and computations over long-term dependencies.

However, this requires to hold inputs and/or past states in memory in order to be considered for attentive computations, a process that typically scales quadratically with sequence length. As for recurrent networks, a promising solution to this has been the use of sparse attention (see e.g. SAB [**35**]), leading to models with varying degrees of reliance on both recurrence and self-attention. However most of these models only delay computational issues by a constant factor and are thus still scaling quadratically with sequence length.

In this paper, we are pushing the idea of sparse attention in RNNs further, by asking: *how can we sparsify attention in RNNs, in order to maximally reduce computational complexity and memory usage, while simultaneously maintaining good gradient propagation over large time scales?*

In Section 2, we give a brief outline of related cognitive processes and neural network mechanisms. In Section 3, we derive asymptotic guarantees for gradient propagation in self-attentive recurrent networks. Building on these results in Section 4, we showcase a simple *relevancy screening mechanism* that aims to efficiently consolidate relevant memory, reducing the size of the computational graph from quadratic to linear in sequence length. Finally, in Section 5, we compare various recurrent and attention models with our proposed relevancy screening mechanism on a series of simple numerical experiments, while, in Section 6, we analyze their gradient propagation properties together with their GPU usage.

2. Background

To perform complex tasks, our brains rely on mechanisms to encode and retrieve information to and from memory [**75, 56**]. In contrast, standard RNNs follow rigid sequential dynamics as they are parametric i.e with a fixed-size state vector. Self-attention methods can overcome this limitation by giving access to previous past states for computing the next state. For the sake of the discussion, we call such RNNs, which are augmented by the memory of the past states as *semi-parametric* RNNs. The use of soft-attention [**5**] in such models has improved performance on many tasks such as reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations [**51, 44, 54, 72**] as well as in the self-supervised training of extremely large language models [**16, 55**] due to their ability to handle long-term dependencies.

Intriguingly, the most notable advances in the use of attention is in purely attention-based systems such as the Transformer [**65**], which completely foregoes recurrence and inspired some of the work listed above. While the performance of these systems is impressive, their memory and computation requirements grows quadratically with the total sequence length. To address this issue, many variants that aim to "sparsify" the attention matrix have been proposed. Notably, [**35**] developed the Sparse Attentive Backtracking model (SAB), a self-attentive Long Short-Term Memory network (LSTM) [**59**] that leverages sparsity by selecting

only the top-k states in memory based on an attention score, propagating gradients only to those chosen hidden states. Recently, [**77**] propose to use a similar top-k attention, and [**14**] introduce sparse masks which attends to roughly $\sqrt{n}$ locations in memory, implementing explicit selection methods for Transformers. Reformer models [**37**] replace the dot-product attention by locality-sensitive hashing, changing its complexity from $O(T^2)$ to $O(T)$, where T is the sequence length.

Still, most of these approaches naively sub-sample input streams for memory storage. Our brains on the other hand, seem to select relevant information from the recent past to commit to long term memory based on their relevancy, a process often referred to as memory consolidation [**1**]. Attempts at mimicking this sparse temporal selectivity process has shown great promise in a variety of contexts [**20, 50, 23**], and our work aims to formalize this idea for self-attentive recurrent networks.

3. Theoretical analysis of gradient propagation

In this section, we analyze the influence of self-attention onto gradient propagation in recurrent networks with self-attention. In order to so let us first recall the equations of a recurrent neural network with self-attention. We note that even though we are using "vanilla RNNs" in the formulations of our results, any recurrent network can take its place (see Section 5 where we use LSTMs in the experiments). Let $x_t \in \mathbb{R}^m$ be the input and $h_t \in \mathbb{R}^n$ be the hidden state at time step t, satisfying the update equation for all $t \geq 1$,

$$h_{t+1} = \phi(V s_t + U x_{t+1} + b) \tag{3.1}$$

$$s_t = f(h_t, c_t) \tag{3.2}$$

where ϕ is a non-linearity, $f : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$, $V \in \mathbb{R}^{n\times n}$, $U \in \mathbb{R}^{n\times m}$, $b \in \mathbb{R}^n$ and $c_t = \alpha_{1,t} h_1 + \alpha_{2,t} h_2 + \ldots + \alpha_{t,t} h_t$ with $\alpha_{i,t} := \frac{\exp(e_{i,t})}{\sum_{j=1}^{t} \exp(e_{j,t})}$ and $e_{i,t} := a(s_{t-1}, h_i)$, where $a : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ is the attention *alignment function*. Throughout, we assume training is done via gradient descent of a cost function L using back-propagation.

Oftentimes, one uses $s_t = f(h_t, c_t) = h_t + c_t$ (but concatenation would be more general), and for all $t > 1$ and $1 \leq j \leq t$, $a(s_{t-1}, h_j) = v_a^T \cdot \tanh(W_a \cdot s_{t-1} + U_a \cdot h_j)$, where $v_a \in \mathbb{R}^n$, and $W_a, U_a \in \mathbb{R}^{n\times n}$. The latter choice for alignment function is sometimes referred to as "additive self-attention" and was used in the original paper [**5**]. Let us emphasize that the results presented in this section hold independently of the choice of the alignment function as we will discuss later in this section.

3.1. Preliminaries

Our goal in this section is to establish formal propagation rules for a system where multiple paths of signal propagation are possible. We would like to understand the relationship between skip connections (those coming from self-attention) and recurrent connections, as well as how the interplay between the two leads to good gradient propagation. In order to achieve this, we seek to analyze the asymptotic behaviour of $\|\nabla_{h_t} L\| = \| \left(\frac{ds_T}{dh_t}\right)^T \nabla_{s_T} L\|$, as $T \to \infty$. We accomplish this by decomposing $\nabla_{h_t} L$ with respect to all possible gradient backpropagation paths, or in other words, by decomposing $\frac{ds_T}{dh_t}$ into sums of products of Jacobian matrices corresponding to those gradient paths, using Proposition 2.

Proposition 1. *For all $t \geq 1$, $k \geq j \geq 0$, $k' \geq 0$, let $E^{(t)}_{k'} = \frac{\partial s_{t+k'}}{\partial h_t}$, and $F^{(t)}_{k+1,j} = \frac{\partial s_{t+k+1}}{\partial h_{t+j+1}} \cdot J_{t+j} + 1_{j=k} \cdot \frac{\partial s_{t+k+1}}{\partial s_{t+k}}$, with J_{t+j} the Jacobian matrix $\frac{dh_{t+j+1}}{ds_{t+j}}$. Then, we have*

$$\frac{ds_{t+k}}{dh_t} = \sum_{s=0}^{k} \bar{\xi}^{(t)}_{0:k}(s) \tag{3.3}$$

where for all $s \geq 1$, $\bar{\xi}^{(t)}_{0:k}(s) = \sum_{0 \leq i_1 < \ldots < i_s < k} F^{(t)}_{k,i_s} \cdot F^{(t)}_{i_s,i_{s-1}} \cdot \ldots \cdot F^{(t)}_{i_2,i_1} \cdot E^{(t)}_{i_1}$ and where $\bar{\xi}^{(t)}_{0:k}(0) = E^{(t)}_k$. (Proof in Appendix 8.2, Proposition 2)

Here, each term $F^{(t)}_{k,i_s} \cdot F^{(t)}_{i_s,i_{s-1}} \cdot \ldots \cdot F^{(t)}_{i_2,i_1} \cdot E^{(t)}_{i_1}$ corresponds to exactly one gradient path involving exactly $s+1$ skip connections going from t to $t+k$, via the s hidden states $h_{t+i_s+1}, \ldots, h_{t+i_1+1}$. In particular, $\bar{\xi}^{(t)}_{0:k}(s)$ is the sum of all terms containing exactly s Jacobian matrices J, and thus the larger s is, the more $\bar{\xi}^{(t)}_{0:k}(s)$ is prone to vanishing.

Intuition: In order to find paths that are not vanishing as $T \to \infty$, we want to find gradient paths with: **(i)** a bounded path length s so that the number of Jacobian matrices involved in the product is limited. **(ii)** attention scores that are sufficiently bounded away from 0, so that the resulting product of attention scores is sufficiently bounded away from 0 as well. In order to see how exactly the attention weights come into play via matrices E and F, we refer to Proposition 3 from Appendix 8.2.

Defintions: Let us fix an integer $t \geq 1$, an integer $s \in \{1,2,\ldots,T-t\}$, and an ordered set of indices $i_1,i_2,\ldots,i_s \in \{0,1,\ldots,T-t-1\}$, verifying $i_1 \leq i_2 \leq \ldots \leq i_s$.

- For sequences $\{g(T)\}_{T\geq 1}$ and $\{f(T)\}_{T\geq 1}$, we say that $f(T) = \Omega(g(T))$ if there exists positive constants c and T_0 such that $f(T) \geq c \cdot g(T)$ for all $T \geq T_0$.
- At time t, we call a past hidden state h_i a *relevant event* if the attention weight $\alpha_{i,t}$ is sufficiently bounded away from zero.
- We call the s-tuple $(i_1,i_2,\ldots,i_s)$ a *dependency chain γ of depth s*, as it induces a gradient backpropagation path going via the s hidden states $h_{t+i_s+1}, \ldots, h_{t+i_1+1}$.
- We call *dependency depth* the *smallest* depth among all dependency chains where the product of the corresponding attention scores is $\Omega(1)$ as $T \to \infty$.

The central message is that *if the dependency depth is bounded from above and sufficiently small, then we mitigate gradient vanishing.* As we see below, task structure introduces different ways in which this may happen. We now present a formal treatment for specific cases, and lay the groundwork to take advantage of this structure during learning.

3.2. Uniform relevance case

Suppose each state has equal relevance in some task. What can be said about gradient propagation? This translates to having each attention weight $\alpha_{i,t} = 1/t$ for all $t \geq i \geq 1$. We then have dependency chains of depth 1 but with vanishing rate $\Omega(1/T)$, as formalized in the following theorem (cf. 8.3)

Theorem 4. *Let h_t be the hidden state at time t of a vanilla RNN with uniform attention, under mild assumptions on the connectivity matrix V, and trained with respect to a loss L, then if T is the total sequence length, we have*

$$\|\nabla_{h_t} L\| = \Omega(1/T) \tag{3.4}$$

as $T \to \infty$. (proof in Appendix 8.3, Theorem 6)

This corresponds to the case where all past events contribute equal error signals. We also note that this result is independent of the choice of the alignment function a (cf. Remark 8 in the Appendix 8.3).

Intuition: As a "worst case scenario" Theorem 4 reveals the true trade-off of early self-attentive recurrent networks [**5**]. On one hand, the lower bound obtained on gradient norm is substantially better than in a vanilla RNN without attention, where vanishing happens at an exponential rate, as opposed to a polynomial one here. On the other, this situation fulfills the quadratic scaling bounds for computations, and requires that all events be held in memory. Many inputs and tasks do not call for uniform attention and naturally lend themselves to sparse dependency paths for computation. The next case treats this situation.

3.3. Sparse relevance case with bounded dependency depth

Now let us look at a more realistic case where only a sparse subset of past states are relevant for the task at hand, and the gradient needs to access those states efficiently for good learning. Figure 12 illustrates this scenario by showing the attention scores for two input examples computed by a simple self-attentive model [**5**], trained on Copy and Denoise tasks respectively (see Section 5). This structure introduces the possibility to impose sparsity in the computational graph, and to limit memory use. With these constraints in mind, the goal is to engineer dependency chains that enable best gradient propagation between these relevant events.

Notation: We consider a κ-sparse attention mechanism of dependency depth d.

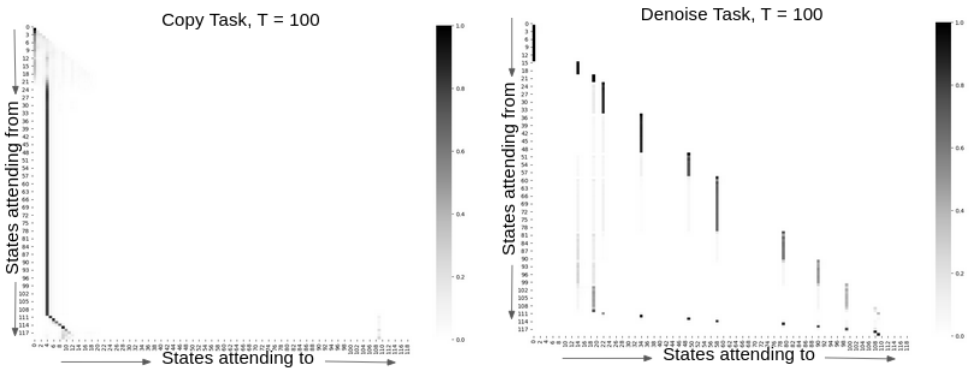

Fig. 12. *Magnitude of attention weights between states in a trained, fully recurrent and fully attentive model* ([**5**]). Each pixel in the lower triangle corresponds to the attention weight of the skip connection departing from the state marked on the y-axis to the state marked on the x-axis. Left shows Copy task, right shows Denoise task. Task details in Section 5

- *Sparsity coefficient*: $\kappa \geq 1$. Borrowing from the SAB model [**35**], at each time step, attention is allowed at most κ relevant events from the past. That is, for any t there are at most κ indices i such that $\alpha_{i,t} \neq 0$, which gives rise to a sparse temporal segmentation via the most relevant events.
- *Maximal dependency depth*: d. This is the maximal dependency depth across all time steps t.

Theorem 5. *Let h_t be the hidden state at time t of a vanilla RNN with κ-sparse uniform attention mechanism of maximal dependency depth d, and under mild assumptions on the connectivity matrix V, then*

$$\|\nabla_{h_t} L\| = \Omega(1/\kappa^d) \tag{3.5}$$

as $T \to \infty$. (proof in Appendix 8.4, Theorem 7)

Similarly to uniform case, Theorem 5 is independent of the choice of the alignment function a (cf. remark 19 in the appendix).

Intuition: If κ and d are assumed to be bounded, we have $\Omega(1/\kappa^d) = \Omega(1)$ as $T \to \infty$, and thus we mitigate gradient vanishing. Furthermore, notice the dependency depth d affects the lower bound exponentially, while κ affects it polynomially. In other words, the number of relevant events attended to at each time step contributes far less to gradient vanishing than the number of events in the longest dependency chain. Theorem 5 outlines the tradeoff between computational complexity as $T \to \infty$ and gradient propagation when balancing attention and recurrence.

4. Relevancy screening mechanism

Equipped with the results from the previous section, we wish to refine heuristics that strike a balance between good gradient propagation and computational/memory complexity. Building on the SAB model [**35**], we remark that although sparse attention attends to the top-κ events at any point in time, attention scores must be computed on all events stored in memory to extract the κ best ones. Thus, the resource bottleneck is not controlled by κ, but rather by the number of stored events in memory. In SAB, there is a naive attempt to control this number by only recording network states at each 10 time steps. However, this reduces the size of the computational graph only by a constant factor, but retains $O(T^2)$ complexity. In contrast, Theorem 5 tells us that the only important events to conserve for good gradient propagation are the *relevant* ones or the ones likely to receive high attention scores (also see Remark 6 in Appendix 8.2). Thus, we propose to reduce complexity while maintaining good gradient propagation by selectively storing events that are predicted to be relevant in the future, using a *relevancy screening mechanism*.

Algorithm 2 Relevancy Screening

procedure: RelRNN($\mathbf{s}_{t-1}, \mathbf{x}_t$)
Require: Previous macro-state - $\mathbf{s}_{t-1}$
Require: Input - $\mathbf{x}_t$, $\nu > 0$, $\rho > 0$
Require: Short-term buffer $s_{t-1}^{(i)} \in S_{t-1}$
Require: Relevant set $r_{t-1}^{(i)} \in R_{t-1}$
$h_t \leftarrow \phi(V\mathbf{s}_{t-1} + U\mathbf{x}_t + b)$
$S_t = S_{t-1}$.add(h_t)
if $t - \nu > 0$ **then**
    $S_t = S_t$.remove($h_{t-\nu}$)
if $t - \rho > 0$ **and** $C(t-\rho) = True$ **then**
    $R_t = R_{t-1}$.replaceWith($h_{t-\rho}$)
$M_t = [S_t, R_t]$
for all $m^{(i)} \in M_t$ **do**
    $\tilde{z}^{(i)} \leftarrow v_a^T \cdot \tanh\left(W_a\mathbf{s}_{t-1} + U_a m^{(i)}\right)$
$z \leftarrow \text{softmax}(\tilde{z})$
$\mathbf{s}_t = h_t + \sum_i z^{(i)} m^{(i)}$
return $\mathbf{s}_t$

The idea is simple: devise a screening function $C(i)$ which estimates the future relevance of h_i, and store selected events in a *relevant set* $R_t = \{h_i | i < t \wedge C(i) = True\}$ for future attention. In principle, one can explicitly control how R_t grows with t, thus mitigating the complexity scaling outlined above. Here, $C(i)$ could take many forms, the best of which depends on task structure. In what follows, we present an example screening mechanism

meant to showcase the lessons learned from Theorem 5, but we refer the interested reader to Section 7 for further possibilities.

We take inspiration from memory consolidation principles in human cognition [**1**], which defines the transfer of events from short-term to long-term memory. We remark that for some tasks such as those depicted in Figure 12, relevance varies very little across time. To implement relevancy screening for such tasks, at every time step t we attend to two subsets of the past hidden states. We call the first subset a *short-term buffer* $S_t = \{h_{t-\nu}, h_{t-\nu+1}, .., h_{t-1}\}$ which consists of the hidden states of the last ν time steps, while the second subset is the relevant set R_t. We compute the *relevance score* at time step i, $\beta(i) = \sum_{j=i}^{i+\nu-1} \alpha_{i,j}$, measuring the integrated attention scores over our short-term buffer S_t. More precisely, $C(i)$ is satisfied if $\beta(i)$ is part of the top ρ relevance scores when compared to all previously observed hidden states, where ρ is a fixed hyper-parameter satisfying $\rho \geq |R_t|$ for all t. The pseudo-code in Algorithm 2 describes the screening mechanisms and the interaction between the short-term buffer S_t and a finite size relevant set R_t. '.replaceWith()' is a function replacing the hidden state with the lowest relevance score by the hidden state in the argument.

We note that the sets S_t and R_t give rise to a sparse attention mechanism with sparsity coefficient κ satisfying $\kappa = \nu + \rho \geq |S_t| + |R_t|$. Hence, memory complexity is constant while the $O(T^2)$ bottleneck of computational complexity is replaced by $O((\rho + \nu) \cdot T) = O(T)$. Lastly, applying Theorem 5, we get the following guarantee for all $t \geq 0$: $\|\nabla_{h_t} L\| = \Omega(1/(\rho+\nu)^d)$ as $T \to \infty$. Thus the choices of ν and ρ not only directly impact computational complexity and gradient propagation, but also indirectly influence gradient propagation via the implicit effect of $\kappa = \nu + \rho$ on d as already discussed in Section 3.

5. Experiments

Before describing experiments, we make a few remarks. First, we stress that Relevancy Screening can be applied to any semi-parametric attentive model but we refer to the version presented below, which uses an RNN/LSTM base, as RelRNN/RelLSTM ("*Relevance RNN /LSTM*"). Second, our objective is not to find state-of-the-art performance but to highlight the advantages of event relevancy and selective sparsity. Finally, we note that relevancy-based sparsity does not necessarily improve performance over fully attentive models, but rather allows efficient and scalable usage. As we show below, RelRNN and RelLSTM perform almost identically to other self-attentive recurrent models (e.g. [**5, 35**]) on simple tasks, but use considerably less memory and compute complexity. In what follows, we denote MemRNN/MemLSTM for vanilla self-attention RNN/LSTM as defined in [**5**]. We consider three task categories to highlight distinct impacts of selective attention and recurrence. The first category specifies tasks with sparse dependency chains, the second one those with dense

T	100	200	400	2000	5000
orth-RNN	99%	4%	16%	10%	0%
expRNN	100%	86%	73%	58%	50%
MemRNN	99%	99%	99%	92%	OOM
RelRNN	100%	99%	99%	99%	99%
LSTM	99%	64%	48%	19%	14%
h-detach	100%	91%	77%	51%	42%
SAB	99%	95%	95%	95%	95%
RelLSTM	100%	99%	99%	99%	99%

Table 4. Results for Transfer Copy task.

temporal dependencies, and the third includes a variety of reinforcement learning (RL) tasks. All implementation details and hyper-parameters can be found in the Appendix 8.6.

5.1. Tasks with sparse dependency chains

A good stereotypical task type that captures sparse sequences of important events are memorization tasks. Here, the network has to memorize a sequence of relevant characters presented among several non-relevant ones, store it for some given time delay and output them in the same order as they were read towards the end of the sequence.

Copy task [**29**]: The characters to be copied are presented in the first 10 time steps, then must be outputted after a long delay of T time steps (see full description in [**3**]). Thus, all the *relevant events* occur in the first 10 time steps. This can be corroborated by the attention score found in Figure 12 which was generated using full self-attention. [**25**] show that orthogonal RNNs (orth-RNN) provide an optimal solution. We also consider expRNN [**12**] which does optimization in the unitary space and is so far the best purely performing recurrent model for large time delays for this task.

Table 8 (Appendix 8.6) reports test performances of orth-RNN, expRNN, MemRNN, SAB, RelRNN and RelLSTM for $T = \{100, 200, 300, 500, 1000, 2000\}$ on the Copy Task. We find that orth-RNN solves this task up to $T = 500$, but that accuracy decays beyond that point, similarly to LSTM. RelRNN, RelLSTM, SAB and expRNN perfectly solve this task with **100**% accuracy for all T, and learn much faster than orth-RNN. MemRNN solves this task until $T = 100$ but overflows memory (OOM) afterwards.

Transfer Copy task: An important advantage of sparse attentive recurrent models such as RelRNN is that of generalization. This is illustrated by the Transfer Copy scores [**29**] where models are trained on Copy task for $T = 100$ and evaluated for $T > 100$. Table 4 shows results for the models listed above, in addition to h-detach [**4**], an LSTM-based model with improved gradient propagation. Importantly, where purely recurrent networks performed well on the original task, all fail to transfer, with discrepancy growing with T. As expected,

T	100	300	500	1000	2000
orth-RNN	90%	71%	61%	29%	3%
expRNN	34%	25%	20%	16%	11%
MemRNN	99%	99%	99%	99%	OOM
RelRNN	100%	99%	99%	99%	99%
LSTM	82%	41%	33%	21%	15%
GORU	92%	93%	91%	93%	73%
SAB	99%	99%	99%	99%	99%
RelLSTM	100%	99%	99%	99%	99%

Table 5. Results for Denoise task.

MemRNN and SAB keep good performance but RelRNN outperforms them, with almost perfect performance for all T. While both SAB and RelRNN use sparse memory storage and retrieval, the distinguishing factor is RelRNN's use of relevancy screening, indicating it's importance for transfer. The performance of RelLSTM on Transfer Copy is exactly the same as RelRNN.

Denoise task [**31**]: This generalizes the Copy task as the symbols that need to be copied are now randomly distributed among the T time steps, requiring the model to selectively pick the inputs that need to be copied. We test our method against all the previously mentioned models in addition to GORU [**31**] for various values of T (Table 5). RelLSTM performs exactly as RelRNN and again, we see RelRNN maintain complete performance across all T values, outperforming all purely recurrent models. MemRNN performs as RelRNN/RelLSTM but fails to train due to memory overflow beyong $T = 500$.

5.2. Tasks with dense temporal dependencies

In contrast to sparse information found in the tasks above, we now illustrate RelRNN and RelLSTM's performance on tasks with densely distributed information on long sequences.

Here, we perform tests on pMNIST [**39**], a variant of MNIST [**40**] where pixels are fed sequentially in a permuted order to the network, as well as character level Penn Tree Bank corpus (PTB) [**47**] where the next letter in a text needs to be predicted. See Table 6 for results. Implementation details and further test data found in Appendix 8.6, including attention heatmaps such as the ones found in Figure 12, showing dense attention for RelRNN in both tasks. We note that gated RNNs such as LSTMs are known to perform well here, and that orthogonal RNNs such as those tested here are also very good. The full attention model (MemRNN) fails to train on the optimization setup used here for both tasks, again due to overflow in memory. The current state of the art in pMNIST is 98.13% achieved by TrellisNet [**6**] and for PTB is a bpc of 1.147 achieved by Recurrent highway networks [**46**].

	PTB Task		pMNIST
Model	BPC	Accuracy	Accuracy
RNN	1.56	66%	90.4%
orth-RNN	1.53	66%	93.4%
expRNN	1.49	68%	96.6%
RelRNN	1.43	69%	92.8%
LSTM	1.36	73%	91.1%
h-detach	-	-	92.3%
SAB	1.37	-	94.2%
RelLSTM	1.36	73%	94.3%

Table 6. PTB and pMNIST results.

Environment	LSTM	MemLSTM	RelLSTM
		Train	
RedBlueDoors-6x6	0.97	0.97	0.97
GoToObject-6x6	0.81	0.85	0.84
MemoryS7	0.96	0.4	0.94
GoToDoor-5x5	0.28	0.17	0.25
Fetch-5x5	0.48	0.42	0.5
DoorKey-5x5	0.94	0.94	0.93
		Test	
RedBlueDoors-8x8	0.95	0.95	0.95
GoToObject-8x8	0.66	0.66	0.74
MemoryS13	0.25	0.24	0.30
GoToDoor-8x8	0.13	0.11	0.15
Fetch-8x8	0.38	0.44	0.45
DoorKey-16x16	0.09	0.31	0.44

Table 7. Average Train and Test Rewards for MiniGrid Reinforcement Learning task. The models were trained on the smaller version of the environment and tested on the larger version to test to generalization of the solution learned.

5.3. MiniGrid reinforcement learning tasks

We next consider a few tasks from MiniGrid [**13**] in the OpenAI gym [**11**] in which an agent must get to certain goal states. We use a partially observed formulation of the task, where the agent only sees a small number of squares ahead of it.

These tasks are difficult to solve with standard RL algorithms, due to (1) the partial observability of the environment and (2) the sparsity of the reward, given that the agent receives a reward only after reaching the goal. We use Proximal Policy Optimization (PPO, [**60**]) along with LSTM, MemLSTM, and RelLSTM as the recurrent modules. All models were each trained for 5000000 steps on each environment. The hyperparameters used for

RelLSTM are $\nu = 5$ and $\rho = 5$. Our goal is to compare generalization of the solutions learned by each model by training on smaller version of an environment and testing it on a larger version. On the *MiniGrid-DoorKey-5x5-v0* environment the average reward for LSTM is 0.94, MemLSTM is 0.94 and RelLSTM is 0.93. On transferring the learned solution to the *16x16* version of that environment the average reward for LSTM is 0.09, MemLSTM is 0.31 and RelLSTM is **0.44**. As illustrated in Table 7, we find that transfer scores for RelLSTM are much higher than for MemLSTM across several environments.

6. Analysis

In this section we analyze the maximal GPU usage and gradient norm of $\|\nabla_{h_t} L\|$ across time t for the Denoise Task. All the models were run using a NVIDIA TitanXP GPU and their peak usage was recorded in order to quantify the amount of computational resources used for each of them. We varied sequence length T from 200 to 2000 in order to measure the trend in the usage. To measure propagating gradients as a function of t, we trained models on $T = 1000$ and computed $\log \|\nabla_{h_t} L\|$.

As illustrated in Figure 13 (center), we confirm MemRNN scales quadratically with T, same as SAB which shows an improvement but only by a constant factor. We also confirm that RelLSTM scales linearly with T similar to RNN and LSTM. Figure 13 (left) shows that the gradient norms for RNN explode and for LSTM vanish as t increases. The gradient norms of all attention models were stable, as expected from the results of Section 3. To better visualize the interplay between gradient norm and GPU usage, Figure 13 (right) shows the final averaged log gradient norm against Max GPU usage for different times $T = \{400, 600, 800\}$. As expected, purely recurrent models (RNN, LSTM) show very little GPU usage differences across distinct T values, while their performance and gradients degrade with increasing t. Note that the RNN's gradients explode while the LSTM's vanish, both exponentially in t. Standard self attentive models (MemRNN, SAB) on the other hand, show opposite trends, with stable gradients but GPU usage quadratically increasing in T. As expected from Theorem 2 (Section 3), RelLSTM shows both stable gradients and stable GPU usage. The measurements for both GPU usage and gradient norm are identical for both RelLSTM and RelRNN.

The optimal trade-off between memory usage and good gradient propagation achieved by RelLSTM highlights the importance of a dynamic memory that attempts to predict relevancy in order to only store exactly those events that help with learning. We note the Denoise task has a small number of relevant events and that not all tasks share this structure. Nevertheless, this experiment highlights how important resource gains can be made by shifting efforts from offsetting memory growth by a constant factor, to a relevancy screening method.

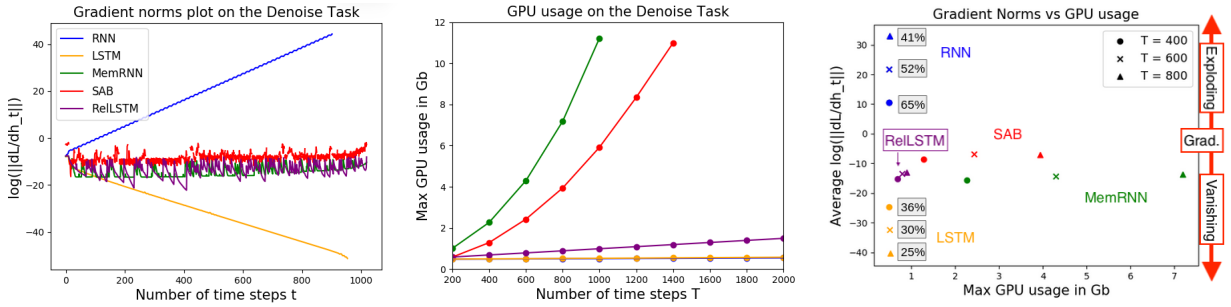

Fig. 13. (Left) gradient norm plots of $\|\nabla_{h_t} L\|$ in log scale after training for Denoise Task with t ranging from 0 (latest time step) to 1000 (furthest time step).**(Center)** Maximal GPU usage as a function of total sequence length T.**(Right)** Mean log gradient norm v.s. Max GPU usage for $T = 400,600,800$. Model testing accuracy is 100% unless indicated by marker label (see Table 5).

7. Conclusion & Discussion

Our main contribution is a formal analysis of gradient propagation in self-attention RNNs, from which we derive two quantities that are governing gradient propagation: sparsity and dependency depth. Meanwhile we identify event relevancy as a key concept to efficiently scale attentive systems to very long sequential computations. This is illustrated via a *Relevancy Screening Mechanism*, inspired by the cognitive process of memory consolidation in the brain, that efficiently selects network states, called relevant events, to be committed to long-term memory based on a screening heuristic operating on a fixed-size short-term memory buffer. We showcase the benefits of this mechanism in an attentive RNN and LSTM which we call RelRNN and RelLSTM respectively, using simple but illustrative numerical experiments, and demonstrate the optimal trade-off between memory usage and good gradient propagation it achieves.

As outlined in Sections 3 and 4, this trade-off is a reflection of the task-specific balance between sparsity and dependency depth parameters. While our proposed relevancy screening mechanism exploits "local" attention scores (measured while events are in short-term memory buffer), we acknowledge other types of relevancy could be formulated with heuristics better suited to distinct environments. For instance, promising directions include leveraging predictive coding principles to select "surprising events", or neural networks could be used to learn the screening function $C(i)$ in an end-to-end fashion.

8. Appendix

8.1. Notational convention

In this paper, we use the notation $\frac{df}{dx}$ to denote the total derivative of f with respect to x, and $\frac{\partial f}{\partial x}$ to denote the partial derivative of f with respect to x.

If we assume $f : \mathbb{R}^n \to \mathbb{R}^m$, and $x \in \mathbb{R}^n$, then $\frac{df}{dx}$ denotes the Jacobian matrix J_f such that

$$(J_f)_{ij} = \frac{df_i}{dx_j} \tag{8.1}$$

In particular, with this notation, we have that if a function $L : \mathbb{R}^m \to \mathbb{R}$, and $y \in \mathbb{R}^m$ then $\frac{dL}{dy}$ is a row vector, while the conventional notation for $\nabla_y L$ indicates a column vector. In other words, $(\nabla_y L)^T = \frac{dL}{dy}$. Hence if L is a function of $f(x)$, then

$$\frac{dL}{dx} = \frac{dL}{df} \cdot \frac{df}{dx} \tag{8.2}$$

while

$$\nabla_x L = \left(\frac{df}{dx}\right)^T \cdot \nabla_{f(x)} L = J_f^T \cdot \nabla_{f(x)} L \tag{8.3}$$

Similarly, we have that $\frac{\partial L}{\partial y}$ is a row vector.

8.2. Preliminary results

Let

$$s_t = \psi_t(h_1,h_2,\ldots,h_t,s_{t-1}) \tag{8.4}$$

where

$$h_{i+1} = \phi(V s_i + U x_{i+1} + b) \tag{8.5}$$

Lemma 2. *For all $t,k \geq 0$, we have*

$$\frac{ds_{t+k+1}}{dh_t} = \frac{\partial s_{t+k+1}}{\partial h_t} + \left(\sum_{j=0}^{k} \frac{\partial s_{t+k+1}}{\partial h_{t+j+1}} \frac{dh_{t+j+1}}{dh_t}\right) + \frac{\partial s_{t+k+1}}{\partial s_{t+k}} \frac{ds_{t+k}}{dh_t} \tag{8.6}$$

Proof. Follows directly from the following multivariable chain rule: if

$$g(t) = f(g_1(t),g_2(t),\ldots,g_n(t)) \tag{8.7}$$

then

$$\frac{dg}{dt} = \sum_{i=1}^{n} \frac{\partial f}{\partial g_i} \frac{dg_i}{dt} \tag{8.8}$$

□

Lemma 3. *If we further denote the Jacobian matrix* $J_k = \frac{\partial s_{k+1}}{\partial h_k}$*, then we get that for all* $t,k \geq 0$*, we have*

$$\frac{ds_{t+k+1}}{dh_t} = \frac{\partial s_{t+k+1}}{\partial h_t} + \sum_{j=0}^{k} \left(\frac{\partial s_{t+k+1}}{\partial h_{t+j+1}} \cdot J_{t+j} + 1_{j=k} \cdot \frac{\partial s_{t+k+1}}{\partial s_{t+k}} \right) \cdot \frac{ds_{t+j}}{dh_t} \tag{8.9}$$

Proof. Follows directly from the observation that

$$\frac{dh_{t+j+1}}{dh_t} = \frac{\partial h_{t+j+1}}{\partial s_{t+j}} \frac{ds_{t+j}}{dh_t} = J_{t+j} \cdot \frac{ds_{t+j}}{dh_t} \tag{8.10}$$

□

Remark 1. *Let us denote*

$$C_{k+1}^{(t)} = \frac{ds_{t+k+1}}{dh_t} \tag{8.11}$$

$$E_{k+1}^{(t)} = \frac{\partial s_{t+k+1}}{\partial h_t} \tag{8.12}$$

and

$$F_{k+1,j}^{(t)} = \frac{\partial s_{t+k+1}}{\partial h_{t+j+1}} \cdot J_{t+j} + 1_{j=k} \cdot \frac{\partial s_{t+k+1}}{\partial s_{t+k}} \tag{8.13}$$

and thus the recursion formula in Lemma 3 rewrites as

$$C_{k+1}^{(t)} = E_{k+1}^{(t)} + \sum_{j=0}^{k} F_{k+1,j}^{(t)} \cdot C_j^{(t)} \tag{8.14}$$

The next two results highlight how to solve this recursion.

Lemma 4. *Let* $C_i, E_i, F_{i,j} \in \mathbb{R}^{n \times n}$ *such that for all* $k \geq 0$*, we have*

$$C_{k+1} = E_{k+1} + \sum_{j=0}^{k} F_{k+1,j} \cdot C_j \tag{8.15}$$

Then for all $k \geq 1$*, we have*

$$C_k = \xi_{0:k} C_0 + \sum_{r=1}^{k} \xi_{r:k} E_r \tag{8.16}$$

where

$$\xi_{r:k} = \sum_{s=1}^{k-r} \xi_{r:k}(s) \tag{8.17}$$

with

$$\xi_{r:k}(s) = \sum_{r=i_1<\ldots<i_{s+1}=k} F_{i_{s+1},i_s} \cdot F_{i_{s-1},i_{s-2}} \cdot \ldots \cdot F_{i_2,i_1} \tag{8.18}$$

and $\xi_{k:k} = Id$.

PROOF. Let us prove the statement by induction on $k \geq 1$.
For $k = 1$, we have

$$C_1 = E_1 + F_{1,0}C_0 = \xi_{1:1}E_1 + \xi_{0:1}C_0 \tag{8.19}$$

Now let us assume the statement to be true for k, then we get

$$C_{k+1} = E_{k+1} + \sum_{j=0}^{k} F_{k+1,j} \cdot \left(\xi_{0:j}C_0 + \sum_{r=1}^{j} \xi_{r:j}E_r \right) \tag{8.20}$$

$$= E_{k+1} + \left(\sum_{j=0}^{k} F_{k+1,j} \cdot \xi_{0:j} \right) \cdot C_0 + \sum_{j=0}^{k} \sum_{r=1}^{j} F_{k+1,j}\xi_{r:j}E_r \tag{8.21}$$

$$= E_{k+1} + \xi_{0:k+1}C_0 + \sum_{r=1}^{k} \left(\sum_{j=r}^{k} F_{k+1,j}\xi_{r:j} \right) \cdot E_r \tag{8.22}$$

$$= \xi_{k+1:k+1}E_{k+1} + \xi_{0:k+1}C_0 + \sum_{r=1}^{k} \xi_{r:k+1}E_r \tag{8.23}$$

$$= \xi_{0:k+1}C_0 + \sum_{r=1}^{k+1} \xi_{r:k+1}E_r \tag{8.24}$$

$$\tag{8.25}$$

□

Lemma 5. *If we further assume that* $C_0 = E_0$*, then we have for all* $k \geq 1$

$$C_k = E_k + \sum_{s=1}^{k} \sum_{q=s}^{k} \xi_{k-q:k}(s)E_{k-q} \tag{8.26}$$

PROOF. Using the previous lemma, we get

$$C_k = E_k + \sum_{s'=1}^{k} \xi_{0:k}(s')C_0 + \sum_{r=1}^{k-1} \sum_{s=1}^{k-r} \xi_{r:k}(s)E_r \tag{8.27}$$

Using the assumption $C_0 = E_0$, we get

$$C_k = E_k + \sum_{s'=1}^{k} \xi_{0:k}(s')E_0 + \sum_{r=1}^{k-1}\sum_{s=1}^{k-r} \xi_{r:k}(s)E_r \tag{8.28}$$

$$= E_k + \sum_{r=0}^{k-1}\sum_{s=1}^{k-r} \xi_{r:k}(s)E_r \tag{8.29}$$

$$\tag{8.30}$$

Now let us put $q = k - r$, we get

$$C_k = E_k + \sum_{q=1}^{k}\sum_{s=1}^{q} \xi_{k-q:k}(s)E_{k-q} \tag{8.31}$$

$$= E_k + \sum_{s=1}^{k}\sum_{q=s}^{k} \xi_{k-q:k}(s)E_{k-q} \tag{8.32}$$

$$\tag{8.33}$$

□

Remark 2. *First, note that Lemma 5 applies here, since $C_0^{(t)} = E_0^{(t)}$, and thus*

$$C_k^{(t)} = E_k^{(t)} + \sum_{s=1}^{k}\sum_{q=s}^{k} \xi_{k-q:k}^{(t)}(s)E_{k-q}^{(t)} \tag{8.34}$$

The idea of Lemma 5 was to regroup all terms with the same number of F factors (where each F contains a Jacobian matrix J_k which contains the connectivity matrix V of the recurrent net). One could roughly perceive the term

$$\sum_{q=s}^{k} \xi_{k-q:k}^{(t)}(s)E_{k-q}^{(t)} \tag{8.35}$$

as being the term of degree s for $s = 1,2,\ldots,k$ and $E_k^{(t)}$ the term of degree 0. This will allow us to consider the terms C roughly as a polynomial in V and we can look the asymptotic behaviour of each of the coefficients of this polynomial individually. This will then give us a very good understanding on how the distribution of the attention weights are affecting the magnitude of total gradient.

Proposition 2. *For all $t \geq 1$, and all $k \geq 0$, we have that*

$$\frac{ds_{t+k}}{dh_t} = \sum_{s=0}^{k} \bar{\xi}_{o:k}^{(t)}(s) \tag{8.36}$$

where for all $s \geq 1$,

$$\bar{\xi}_{o:k}^{(t)}(s) = \sum_{0 \leq i_1 < \ldots < i_s < k} F_{k,i_s}^{(t)} \cdot F_{i_s,i_{s-1}}^{(t)} \cdot \ldots \cdot F_{i_2,i_1}^{(t)} \cdot E_{i_1}^{(t)} \tag{8.37}$$

and where $\bar{\xi}^{(t)}_{o:k}(0) = E^{(t)}_k$. *With for all* $k \geq 0$ *we have*

$$E^{(t)}_k = \frac{\partial s_{t+k}}{\partial h_t} \tag{8.38}$$

and for all $k \geq j$ *we have*

$$F^{(t)}_{k+1,j} = \frac{\partial s_{t+k+1}}{\partial h_{t+j+1}} \cdot J_{t+j} + 1_{j=k} \cdot \frac{\partial s_{t+k+1}}{\partial s_{t+k}} \tag{8.39}$$

PROOF. Let $t \geq 1$, and recall that we defined $C^{(t)}_k = \frac{ds_{t+k}}{dh_t}$, for all $k \geq 0$.

As already pointed out, we know that $C^{(t)}_0 = E^{(t)}_0$ (thus the claim holds for $k = 0$).

Then by Lemma 5, we know that for all $k \geq 1$ we have

$$C^{(t)}_k = E^{(t)}_k + \sum_{s=1}^{k} \sum_{q=s}^{k} \xi^{(t)}_{k-q:k}(s) E^{(t)}_{k-q} \tag{8.40}$$

$$= \bar{\xi}^{(t)}_{o:k}(0) + \sum_{s=1}^{k} \sum_{q=s}^{k} \sum_{k-q=i_1<\ldots<i_{s+1}=k} F^{(t)}_{k,i_s} \cdot F^{(t)}_{i_s,i_{s-1}} \cdot \ldots \cdot F^{(t)}_{i_2,i_1} \cdot E^{(t)}_{i_1} \tag{8.41}$$

$$= \bar{\xi}^{(t)}_{o:k}(0) + \sum_{s=1}^{k} \sum_{0\leq i_1<\ldots<i_s<k} F^{(t)}_{k,i_s} \cdot F^{(t)}_{i_s,i_{s-1}} \cdot \ldots \cdot F^{(t)}_{i_2,i_1} \cdot E^{(t)}_{i_1} \tag{8.42}$$

$$= \bar{\xi}^{(t)}_{o:k}(0) + \sum_{s=1}^{k} \bar{\xi}^{(t)}_{o:k}(s) \tag{8.43}$$

$$= \sum_{s=0}^{k} \bar{\xi}^{(t)}_{o:k}(s) \tag{8.44}$$

□

Remark 3. *In what follows the main emphasis will be to calculate the* $F^{(t)}_{i,j}$ *and* $E^{(t)}_i$ *terms explicitly, since they are the building blocks of the mentioned polynomials in 2.*

We will assume that

$$s_t = f(h_t, c_t) \tag{8.45}$$

with

$$c_t = \alpha_{1,t} h_1 + \alpha_{2,t} h_2 + \ldots + \alpha_{t,t} h_t \tag{8.46}$$

and

$$\alpha_{j,t} = \frac{\exp(e_{j,t})}{\sum_{i=1}^{t} \exp(e_{i,t})} \tag{8.47}$$

where

$$e_{i,t} = a(s_{t-1},h_i) \tag{8.48}$$

Let us recall that for all $k \geq 0$ we have

$$E_k^{(t)} = \frac{\partial s_{t+k}}{\partial h_t} \tag{8.49}$$

and for all $k \geq j$ we have

$$F_{k+1,j}^{(t)} = \frac{\partial s_{t+k+1}}{\partial h_{t+j+1}} \cdot J_{t+j} + 1_{j=k} \cdot \frac{\partial s_{t+k+1}}{\partial s_{t+k}} \tag{8.50}$$

Lemma 6. *With the assumption of Remark 3, we have that for all $t \geq 2$*

$$\frac{\partial s_t}{\partial s_{t-1}} = \partial_2 f(h_t,c_t) \cdot \left(\sum_{i=1}^{t} \alpha_{i,t} Y_{i,t} \right) \tag{8.51}$$

where $\partial_2 f$ is the the partial derivative of f with respect to the second variable, and where we define

$$Y_{i,t} = h_i \cdot \left(\frac{\partial e_{i,t}}{\partial s_{t-1}} - \sum_{j=1}^{t} \alpha_{j,t} \cdot \frac{\partial e_{j,t}}{\partial s_{t-1}} \right) \tag{8.52}$$

Proof.

$$\begin{aligned}
\frac{\partial s_t}{\partial s_{t-1}} &= \partial_2 f(h_t,c_t) \cdot \frac{\partial c_t}{\partial s_{t-1}} && (8.53)\\
&= \partial_2 f(h_t,c_t) \cdot \left[\sum_{i=1}^{t} h_i \cdot \left(\frac{\partial \alpha_{i,t}}{\partial s_{t-1}} \right) \right] && (8.54)\\
&= \partial_2 f(h_t,c_t) \cdot \left[\sum_{i=1}^{t} h_i \cdot \left(\sum_{j=1}^{t} \frac{\partial \alpha_{i,t}}{\partial e_{j,t}} \cdot \frac{\partial e_{j,t}}{\partial s_{t-1}} \right) \right] && (8.55)\\
&= \partial_2 f(h_t,c_t) \cdot \left[\sum_{i=1}^{t} h_i \cdot \left(\sum_{j=1}^{t} \alpha_{i,t}(1_{i=j} - \alpha_{j,t}) \cdot \frac{\partial e_{j,t}}{\partial s_{t-1}} \right) \right] && (8.56)\\
&= \partial_2 f(h_t,c_t) \cdot \left[\sum_{i=1}^{t} \alpha_{i,t} h_i \left(\frac{\partial e_{i,t}}{\partial s_{t-1}} - \sum_{j=1}^{t} \alpha_{j,t} \frac{\partial e_{j,t}}{\partial s_{t-1}} \right) \right] && (8.57)\\
&= \partial_2 f(h_t,c_t) \cdot \left(\sum_{i=1}^{t} \alpha_{i,t} Y_{i,t} \right) && (8.58)
\end{aligned}$$

□

Lemma 7. *With the assumption of Remark 3, we have that for all $k \geq j$:*

$$\frac{\partial s_k}{\partial h_j} = 1_{k=j} \cdot \partial_1 f(h_k,c_k) + \alpha_{j,k} \partial_2 f(h_k,c_k) \cdot (I + X_{j,k}) \tag{8.59}$$

where $\partial_1 f$ and $\partial_2 f$ are the partial derivatives of f with respect to the first and second variable, respectively, and where we define

$$X_{j,k} = \left(h_j - \sum_{i=1}^{k} h_i \alpha_{i,k} \right) \cdot \frac{\partial e_{j,k}}{\partial h_j} \tag{8.60}$$

Proof.

$$\frac{\partial s_k}{\partial h_j} = 1_{k=j} \cdot \partial_1 f(h_k,c_k) \cdot \frac{\partial h_k}{\partial h_k} + \partial_2 f(h_k,c_k) \cdot \frac{\partial c_k}{\partial h_j} \tag{8.61}$$

$$= 1_{k=j} \cdot \partial_1 f(h_k,c_k) + \partial_2 f(h_k,c_k) \cdot \left[\alpha_{j,k} \cdot I + \sum_{i=1}^{k} h_i \cdot \frac{\partial \alpha_{i,k}}{\partial h_j} \right] \tag{8.62}$$

$$= 1_{k=j} \cdot \partial_1 f(h_k,c_k) + \partial_2 f(h_k,c_k) \cdot \left[\alpha_{j,k} \cdot I + \sum_{i=1}^{k} h_i \cdot \frac{\partial \alpha_{i,k}}{\partial e_{j,k}} \frac{\partial e_{j,k}}{\partial h_j} \right] \tag{8.63}$$

$$= 1_{k=j} \cdot \partial_1 f(h_k,c_k) + \partial_2 f(h_k,c_k) \cdot \left[\alpha_{j,k} \cdot I + \sum_{i=1}^{k} h_i \cdot \alpha_{i,k}(1_{i=j} - \alpha_{j,k}) \frac{\partial e_{j,k}}{\partial h_j} \right] \tag{8.64}$$

$$= 1_{k=j} \cdot \partial_1 f(h_k,c_k) + \partial_2 f(h_k,c_k) \cdot \left[\alpha_{j,k} \cdot I + \left(h_j \alpha_{j,k} - \alpha_{j,k} \sum_{i=1}^{k} h_i \cdot \alpha_{i,k} \right) \frac{\partial e_{j,k}}{\partial h_j} \right] \tag{8.65}$$

$$= 1_{k=j} \cdot \partial_1 f(h_k,c_k) + \alpha_{j,k} \partial_2 f(h_k,c_k) \cdot \left[I + \left(h_j - \sum_{i=1}^{k} h_i \cdot \alpha_{i,k} \right) \frac{\partial e_{j,k}}{\partial h_j} \right] \tag{8.66}$$

$$= 1_{k=j} \cdot \partial_1 f(h_k,c_k) + \alpha_{j,k} \partial_2 f(h_k,c_k) \cdot (I + X_{j,k}) \tag{8.67}$$

□

Corollary 6. *With the assumption of Remark 3, and the notations of lemma 6 and 7, we have for all $k' \geq 0$,*

$$E_{k'}^{(t)} = 1_{k'=0} \partial_1 f(h_t,c_t) + \alpha_{t,t+k'} \partial_2 f(h_{t+k'},c_{t+k'}) \cdot [I + X_{t,t+k'}] \tag{8.68}$$

and for all $k \geq j$,

$$F_{k+1,j}^{(t)} = \alpha_{t+j+1,t+k+1} \cdot \partial_2 f(h_{t+k+1},c_{t+k+1}) \cdot [I + X_{t+j+1,t+k+1}] \cdot J_{t+j} \tag{8.69}$$

$$+ 1_{k=j} \cdot \left(\partial_1 f(h_{t+k+1},c_{t+k+1}) J_{t+j} + \partial_2 f(h_{t+k+1},c_{t+k+1}) \cdot \left[\sum_{i=1}^{t+k+1} \alpha_{i,t+k+1} Y_{i,t+k+1} \right] \right) \tag{8.70}$$

Proof. Applying lemma 7, we get that for all $k \geq 0$,

$$E_{k'}^{(t)} = \frac{\partial s_{t+k'}}{\partial h_t} \tag{8.71}$$

$$= 1_{k'=0} \cdot \partial_1 f(h_t,c_t) + \alpha_{t,t+k'} \cdot \partial_2 f(h_{t+k'},c_{t+k'}) \cdot [I + X_{t,t+k'}] \tag{8.72}$$

$$\tag{8.73}$$

and then by applying lemma 6 and 7, we get that for all $k \geq j$,

$$F_{k+1,j}^{(t)} = \frac{\partial s_{t+k+1}}{\partial h_{t+j+1}} \cdot J_{t+j} + 1_{j=k} \cdot \frac{\partial s_{t+k+1}}{\partial s_{t+k}} \tag{8.74}$$

$$= [1_{k=j} \partial_1 f(h_{t+k+1}, c_{t+k+1}) \tag{8.75}$$

$$+ \alpha_{t+j+1,t+k+1} \partial_2 f(h_{t+k+1}, c_{t+k+1}) \cdot (I + X_{t+j+1,t+k+1})] \cdot J_{t+j} \tag{8.76}$$

$$+ 1_{k=j} \cdot \partial_2 f(h_{t+k+1}, c_{t+k+1}) \cdot \left(\sum_{i=1}^{t+k+1} \alpha_{i,t+k+1} Y_{i,t+k+1} \right) \tag{8.77}$$

$$= \alpha_{t+j+1,t+k+1} \cdot \partial_2 f(h_{t+k+1}, c_{t+k+1}) \cdot [I + X_{t+j+1,t+k+1}] \cdot J_{t+j} \tag{8.78}$$

$$+ 1_{k=j} \cdot \left(\partial_1 f(h_{t+j+1}, c_{t+k+1}) J_{t+j} + \partial_2 f(h_{t+k+1}, c_{t+k+1}) \cdot \left[\sum_{i=1}^{t+k+1} \alpha_{i,t+k+1} Y_{i,t+k+1} \right] \right) \tag{8.79}$$

□

Proposition 3. *We can rewrite for all $k' \geq 0$ and all $k \geq j \geq 0$*

$$E_{k'}^{(t)} = \alpha_{t,t+k'} \cdot \tilde{D}_{k',0}^{(t)} + 1_{k'=0} \tilde{R}_0^{(t)} \tag{8.80}$$

$$F_{k+1,j}^{(t)} = \alpha_{t+j+1,t+k+1} \cdot D_{k+1,j}^{(t)} + 1_{k=j} \cdot R_{k+1}^{(t)} \tag{8.81}$$

where

$$D_{k+1,j+1}^{(t)} = \partial_2 f(h_{t+k+1}, c_{t+k+1}) \cdot [I + X_{t+j+1,t+k+1}] \cdot J_{t+j} \tag{8.82}$$

$$R_{k+1}^{(t)} = \partial_1 f(h_{t+k+1}, c_{t+k+1}) \cdot J_{t+k} + \partial_2 f(h_{t+k+1}, c_{t+k+1}) \cdot [\sum_{i=1}^{t+k+1} \alpha_{i,t+k+1} Y_{i,t+k+1}] \tag{8.83}$$

$$\tilde{D}_{k'}^{(t)} = \partial_2 f(h_{t+k'}, c_{t+k'}) \cdot [I + X_{t,t+k'}] \tag{8.84}$$

$$\tilde{R}_0^{(t)} = \partial_1 f(h_t, c_t) \tag{8.85}$$

while $X_{i,i'}$ and $Y_{i,i'}$ are defined as in lemma 6 and 7.

Proof. Follows straight from Corollary 6. □

Remark 4. *If we are further assuming that*

$$s_t = f(h_t, c_t) = h_t + c_t \tag{8.86}$$

then for all $k \geq 0$, we have

$$E_k^{(t)} = 1_{k=0} \cdot I + \alpha_{t,t+k} \cdot [I + X_{t,t+k}] \tag{8.87}$$

and for all $k \geq j$, we have

$$F^{(t)}_{k+1,j} = \alpha_{t+j+1,t+k+1} \cdot [I + X_{t+j+1,t+k+1}] \cdot J_{t+j} + 1_{k=j} \cdot \left(J_{t+j} + \left[\sum_{i=1}^{t+k+1} \alpha_{i,t+k+1} Y_{i,t+k+1} \right] \right) \tag{8.88}$$

PROOF. This follows directly form corollary 6 and the observation that

$$\partial_1 f(h_t,c_t) = \partial_2 f(h_t,c_t) = I \tag{8.89}$$

□

Remark 5. *If we are further assuming that*

$$e_{j,t} = a(s_{t-1},h_j) = v_a^T \cdot \tanh\left(W_a s_{t-1} + U_a h_j\right) \tag{8.90}$$

as done in [**5**]*, we get that*

$$\frac{\partial e_{j,t}}{\partial h_j} = v_a^T \cdot diag[1 - \tanh^2\left(W_a s_{t-1} + U_a h_j\right)] \cdot U_a \tag{8.91}$$

and

$$\frac{\partial e_{j,t}}{\partial s_{t-1}} = v_a^T \cdot diag[1 - \tanh^2\left(W_a s_{t-1} + U_a h_j\right)] \cdot W_a \tag{8.92}$$

which we can plug into the definitions of $X_{j,k}$ and $Y_{j,k}$ to get explicit expressions for matrices $E^{(t)}_{k'}$ and $F^{(t)}_{k+1,j}$.

Lemma 8. *If, with the assumptions Remark 3, we assume that for all $i,t \geq 1$, we have $e_{i,t} = a(s_{t-1},h_i,\theta)$ depending on some parameter $\theta \in \mathbb{R}^{N \times M}$, then we have*

$$\frac{dL}{d\theta} = \sum_{j,t} \alpha_{j,t} \cdot \frac{dL}{ds_t} \cdot \partial_2 f(h_t,c_t) \cdot h_j \cdot \left[\sum_i (1_{i=j} - \alpha_{i,t}) \cdot \frac{\partial e_{i,t}}{\partial \theta} \right] \tag{8.93}$$

PROOF. If we denote $\theta^{(i,t)}$ to be the parameter for $e_{i,t}$, then we have

$$\frac{dL}{d\theta} = \sum_{i,t} \frac{dL}{d\theta^{(i,t)}} \tag{8.94}$$

$$= \sum_{i,j,t} \frac{dL}{d\alpha_{j,t}} \cdot \frac{\partial \alpha_{j,t}}{\partial e_{i,t}} \cdot \frac{\partial e_{i,t}}{\partial \theta^{(i,t)}} \tag{8.95}$$

$$= \sum_{i,j,t} \alpha_{j,t}(1_{i=j} - \alpha_{i,t}) \cdot \frac{dL}{d\alpha_{j,t}} \cdot \frac{\partial e_{i,t}}{\partial \theta} \tag{8.96}$$

where

$$\frac{dL}{d\alpha_{j,t}} = \frac{dL}{ds_t} \cdot \frac{\partial s_t}{\partial c_t} \cdot \frac{\partial c_t}{\partial \alpha_{j,t}} = \frac{dL}{ds_t} \cdot \partial_2 f(h_t,c_t) \cdot h_j \tag{8.97}$$

Hence

$$\frac{dL}{d\theta} = \sum_{i,j,t} \alpha_{j,t}(1_{i=j} - \alpha_{i,t}) \cdot \frac{dL}{ds_t} \cdot \partial_2 f(h_t,c_t) \cdot h_j \cdot \frac{\partial e_{i,t}}{\partial \theta} \tag{8.98}$$

$$= \sum_{j,t} \alpha_{j,t} \cdot \frac{dL}{ds_t} \cdot \partial_2 f(h_t,c_t) \cdot h_j \cdot \left[\sum_i (1_{i=j} - \alpha_{i,t}) \cdot \frac{\partial e_{i,t}}{\partial \theta} \right] \tag{8.99}$$

□

Lemma 9. *Let us recall that for all $t \geq 0$, we have*

$$h_{t+1} = \phi(\underbrace{Vs_t + Ux_{t+1} + b}_{=a_t}) \tag{8.100}$$

where ϕ is a non-linear activation function, $V \in \mathbb{R}^{n \times n}$, $U \in \mathbb{R}^{n \times m}$ and $b \in \mathbb{R}^n$. Then we have that

$$\left[\frac{dL}{dV}, \frac{dL}{dU}, \frac{dL}{db} \right] = \sum_{t=1}^{T} [s_{t-1},x_t,1] \cdot \frac{dL}{dh_t} \cdot diag(\phi'(a_t)) \tag{8.101}$$

Proof. Let us denote $V^{(t)}, U^{(t)}, b^{(t)}$ the matrices V,U,b of a_{t-1} respectively, then

$$\left[\frac{dL}{dV}, \frac{dL}{dU}, \frac{dL}{db} \right] = \sum_t \left[\frac{dL}{dV^{(t)}}, \frac{dL}{dU^{(t)}}, \frac{dL}{db^{(t)}} \right] \tag{8.102}$$

$$= \sum_t [s_{t-1},x_t,1] \cdot \frac{dL}{da_{t-1}} \tag{8.103}$$

$$= \sum_t [s_{t-1},x_t,1] \cdot \frac{dL}{dh_t} \cdot \frac{dh_t}{da_{t-1}} \tag{8.104}$$

$$= \sum_t [s_{t-1},x_t,1] \cdot \frac{dL}{dh_t} \cdot \mathrm{diag}(\phi'(a_{t-1})) \tag{8.105}$$

□

Remark 6. *Combining the fact that $\frac{dL}{dh_t} = \frac{dL}{ds_T}\frac{ds_T}{dh_t}$, the results from propositions 2 and 3, with lemma 9, we see that attention weights $\alpha_{i,t}$ which are very close to 0, do not contribute to the gradient and the learning of V,U and b.*

Similarly, it follows directly from lemma 8, that attention weights $\alpha_{i,t}$ which are very close to 0, do not contribute to the gradient and the learning of any parameters θ of the alignment function $e_{i,t} = a(s_{t-1},h_i,\theta)$. In case we have an alignment function as in remark 5, these parameters are W_a, U_a and v_a.

If we have the case where one state h_i is such that all attention weights $\alpha_{i,t} \approx 0$ for all $t \geq i$, then we can see that h_i does not contribute to the gradient and learning to any

parameters be it parameters from the recurrence or the alignment function.

In practice we have observed that in the majority of tasks, most states h_i fall in either of two categories:

- *$\alpha_{i,t}$ is sufficiently bounded away from 0 for most $t \geq i$, and thus contributes to learning. This is what we call a "relevant state".*
- *$\alpha_{i,t} \approx 0$ for almost all $t \geq i$, and thus doesn't contribute much to learning, and the gradient can be approximated by assuming $\alpha_{i,t} = 0$ for all $t \geq i$. This is what he call a "non-relevant state".*

This observation is what lead us to the intuition that we can approximate the gradient, by decomposing it via proposition 2, into gradient paths involving only skip connections between "relevant states".

8.3. Uniform attention case

Remark 7. *In this subsection, we are going to assume:*

- *no non-linearity in the hidden-to-hidden connection: $J_t = V$ for all t.*
- *all assumptions from Remark 3.*
- *uniform attention: $\alpha_{i,t} = 1/t$ for all $t \geq 1$.*

8.3.1. Overview.

Remark 8. *Recalling corollary 6, together the main proposition 2 form last section, we can hope to simplify these expressions using the new assumptions from the previous remark 7. Recalling expression from lemma 6 and 7:*

$$X_{j,t} = \left(h_j - \sum_{i=1}^{t} h_i \alpha_{i,t} \right) \cdot \frac{\partial e_{j,t}}{\partial h_j} \tag{8.106}$$

$$= \left(h_j - \frac{1}{t} \sum_{i=1}^{t} h_i \right) \cdot \frac{\partial e_{j,t}}{\partial h_j} \tag{8.107}$$

Hence, for our calculations we are going to assume that $\left(h_j - \frac{1}{t}\sum_{i=1}^{t} h_i\right) \approx 0$*, and thus* $X_{j,t} \approx 0$ *for all* $1 \leq j \leq t$*. Similarly,*

$$\sum_{i=1}^{t} \alpha_{i,t} Y_{i,t} = \sum_{i=1}^{t} \alpha_{i,t} h_i \cdot \left(\frac{\partial e_{i,t}}{\partial s_{t-1}} - \sum_{j=1}^{t} \alpha_{j,t} \cdot \frac{\partial e_{j,t}}{\partial s_{t-1}}\right) \tag{8.108}$$

$$= \frac{1}{t}\sum_{i=1}^{t} h_i \cdot \left(\frac{\partial e_{i,t}}{\partial s_{t-1}} - \sum_{j=1}^{t} \frac{1}{t} \cdot \frac{\partial e_{j,t}}{\partial s_{t-1}}\right) \tag{8.109}$$

$$= \frac{1}{t}\sum_{i=1}^{t} h_i \cdot \frac{\partial e_{i,t}}{\partial s_{t-1}} - \frac{1}{t}\sum_{j=1}^{t} \left(\frac{1}{t}\sum_{i=1}^{t} h_i\right) \cdot \frac{\partial e_{j,t}}{\partial s_{t-1}} \tag{8.110}$$

$$= \frac{1}{t}\sum_{i=1}^{t} h_i \cdot \frac{\partial e_{i,t}}{\partial s_{t-1}} - \frac{1}{t}\sum_{i=1}^{t} \left(\frac{1}{t}\sum_{j=1}^{t} h_j\right) \cdot \frac{\partial e_{i,t}}{\partial s_{t-1}} \tag{8.111}$$

$$= \frac{1}{t}\sum_{i=1}^{t} \left(h_i - \frac{1}{t}\sum_{j=1}^{t} h_j\right) \cdot \frac{\partial e_{i,t}}{\partial s_{t-1}} \tag{8.112}$$

$$\approx 0 \tag{8.113}$$

Recalling the expression from corollary 6 and that $f(h_t,c_t) = h_t + c_t$ *by remark 3, and that* $J_t = V$ *for all* t*, this will give for all* $k' \geq 0$

$$E_{k'}^{(t)} = \left(\frac{1}{t+k'} + 1_{k'=0}\right) \cdot I \tag{8.114}$$

and for all $k \geq j$*, we get*

$$F_{k+1,j}^{(t)} = \left(\frac{1}{t+k+1} + 1_{k=j}\right) \cdot V \tag{8.115}$$

Hence by recalling proposition 2, the main expression of interest becomes

$$\frac{ds_{t+k}}{dh_t} = \sum_{s=0}^{k} \bar{\xi}_{0:k}^{(t)}(s) = \sum_{s=0}^{k} V^s \cdot \chi_{0:k}^{(t)}(s) \tag{8.116}$$

where

$$\chi_{0:k}^{(t)}(s) = \sum_{0 \leq i_1 < \ldots < i_s < k} \left(\frac{1}{t+k} + 1_{k-i_s=1}\right) \cdot \left(\frac{1}{t+i_s} + 1_{i_s - i_{s-1}=1}\right) \cdot \ldots \tag{8.117}$$

$$\ldots \cdot \left(\frac{1}{t+i_2} + 1_{i_2-i_1=1}\right) \cdot \left(\frac{1}{t+i_1} + 1_{i_1=0}\right) \tag{8.118}$$

Remark 9. *The goal is thus to have a good estimation of the terms*

$$\chi_{0:k}^{(t)}(s) \tag{8.119}$$

in order to then find an asymptotic estimation for

$$\frac{ds_{t+k}}{dh_t} = \sum_{s=0}^{k} V^s \cdot \chi_{0:k}^{(t)}(s) \tag{8.120}$$

as $k \to \infty$. In order to do so, we will adopt the following strategy:

Step 1. *Estimate the expression*

$$\omega_{l:k}^{(t)}(s) = \sum_{l \leq i_1 < \ldots < i_s < k} \frac{1}{t+i_s} \cdot \frac{1}{t+i_{s-1}} \cdot \ldots \cdot \frac{1}{t+i_2} \cdot \frac{1}{t+i_1} \tag{8.121}$$

for all $s \geq 1$. This will be done in sub-subsection 8.3.2.

Step2. *Estimate the expression*

$$\theta_{l:k}^{(t)}(s) = \sum_{l \leq i_1 < \ldots < i_s < k} \left(\frac{1}{t+i_s} + 1_{i_s - i_{s-1}=1} \right) \cdot \left(\frac{1}{t+i_{s-1}} + 1_{i_{s-1}-i_{s-2}=1} \right) \cdot \ldots \tag{8.122}$$

$$\ldots \cdot \left(\frac{1}{t+i_2} + 1_{i_2 - i_1 = 1} \right) \cdot \left(\frac{1}{t+i_1} + 1_{i_1 = 0} \right) \tag{8.123}$$

for all $s \geq 1$, because as we will see the expression $\theta_{l:k}^{(t)}(s)$ can be decomposed into $\omega_{l':k'}^{(t)}(s')$ expressions for $s \geq s' \geq 1$. This will be done in sub-subsection 8.3.3.

Step 3. *The final step will consist in putting the results from the two previous sub-subsections together, and getting a final asymptotic estimate for $\frac{ds_{t+k}}{dh_t}$ as $k \to \infty$, by noting that*

$$\chi_{0:k}^{(t)}(s) = \frac{1}{t+k} \cdot \theta_{0:k}^{(t)}(s) + \frac{1}{t+k-1} \cdot \theta_{0:k-1}^{(t)}(s-1) + \ldots \tag{8.124}$$

$$\ldots + \frac{1}{t+k-s+1} \cdot \theta_{0:k-s+1}^{(t)}(1) + \frac{1}{t+k-s} + 1_{k=s} \tag{8.125}$$

This will be treated in sub-subsection 8.3.4.

8.3.2. Estimating ω.

Remark 10. *In this sub-subsection we are going to estimate $\omega_{0:k}^{(t)}(s)$, which is a sum of products of s distinct factors. The idea will be to start from the expression*

$$\left(\frac{1}{t} + \frac{1}{t+1} + \ldots + \frac{1}{t+k-1} \right)^s \tag{8.126}$$

and substract all products containing at least two identical factors, followed by a division by $s!$.

This approach will be similar in spirit to the inclusion-exclusion principle, with the only difference that the desired term will not computed directly, but instead one first establishes a recursive formula using $\omega_{0:k}^{(t)}(s')$ with $s' \leq s$.

Solving this recursive formula will enable us to express $\omega_{0:k}^{(t)}(s)$ *only in terms of* $(\frac{1}{t}+\frac{1}{t+1}+\ldots+\frac{1}{t+k-1})$. *In fact,* $\omega_{0:k}^{(t)}(s)$ *will be a polynomial of degree* s *in* $(\frac{1}{t}+\frac{1}{t+1}+\ldots+\frac{1}{t+k-1})$.

We adopt this approach, because we have a very good estimate for

$$\frac{1}{t}+\frac{1}{t+1}+\ldots+\frac{1}{t+k-1} \tag{8.127}$$

Namely, we know that for all n, *we have*

$$1+\frac{1}{2}+\ldots+\frac{1}{n-1}+\frac{1}{n}=\ln n+\gamma+\varepsilon_n\leq \ln n+1 \tag{8.128}$$

where $\gamma>\frac{1}{2}$ *is the Euler-Mascheroni constant and* ε_n *behaves asymptotically as* $\frac{1}{2n}$. *In other words,*

$$\frac{1}{t}+\frac{1}{t+1}+\ldots+\frac{1}{t+k-1}=\ln\left(\frac{t+k-1}{t-1}\right)+\varepsilon_{t+k-1}-\varepsilon_{t-1} \tag{8.129}$$

$$=\ln\left[\frac{t+k-1}{t-1}\cdot\exp\left(\varepsilon_{t+k-1}-\varepsilon_{t-1}\right)\right] \tag{8.130}$$

$$=\ln\beta_{t-1,t+k-1} \tag{8.131}$$

where $\beta_{l,l'}=\frac{l'}{l}\cdot\exp\left(\varepsilon_{l'}-\varepsilon_l\right)$. *In order to reinforce the intuition here, let us imagine that* $T=t+k$, *then*

$$\ln\beta_{t-1,t+k-1}\sim\ln T \tag{8.132}$$

as $T\to\infty$. *Hence we should expect* $\omega_{0:k}^{(t)}(s)$ *to behave asymptotically as a polynomial of degree* s *in* $\ln T$.

Let us emphasize that we would like to express $\omega_{0:k}^{(t)}(s)$ *with as much precision as possible (i.e. not omitting the monomials in* $\ln T$ *of degree less than* s*), since we would like to later on use this estimate in subsequent steps when summing multiple* $\omega_{0:k}^{(t)}(s)$ *terms over* s.

In order to further ease notation, we will simply write $\omega(s)$ *for* $\omega_{0:k}^{(t)}(s)$, *whenever there is no ambiguity.*

Finally, for this sub-subsection only we will use the following notation

$$S_l=\frac{1}{t^l}+\frac{1}{(t+1)^l}+\ldots+\frac{1}{(t+k-1)^l} \tag{8.133}$$

for all $l\geq 1$, *and keeping in mind that* S_l *converges as* $k\to\infty$, *for all* $l\geq 2$.

Remark 11. *Let us now build a first intuition on how to apply an inclusion-exclusion-like principle in order to calculate $\omega(s)$ for small s.*

For $\underline{s=1}$:

$$\omega(1) = S_1 \tag{8.134}$$

For $\underline{s=2}$:

$$\omega(2) = \frac{1}{2!}\left(S_1^2 - S_2\right) \tag{8.135}$$

Here we expand S_1^2, then substract the sum of products of doubles S_2, followed by a division of $2! = 2$ to divide out the number of permutations.

For $\underline{s=3}$: first we need to substract the sum of products of triples S_3, and then the sum products where exactly two factors are identical $S_2 \cdot \omega(1) - S_3$. The latter appears $\binom{3}{2,1} = \frac{3!}{2!1!} = 3$ times in the expansion of S_1^3. Similarly, we need to divide out the number of permutations $3!$. Hence

$$\omega(3) = \frac{1}{3!}\left[S_1^3 - S_3 - 3\cdot(S_2\cdot\omega(1) - S_3)\right] = \frac{S_1^3}{3!} - \frac{1}{2}S_2\cdot\omega(1) + \frac{1}{3}S_3 \tag{8.136}$$

Let us form now on denote (3) for the sum of products of triples, and (2,1) the sum of products where exactly two factors are the same.

More generally we would denote

$$(j_1,j_2,\ldots,j_k) \tag{8.137}$$

with $j_1 \geq j_2 \geq \ldots \geq j_k \geq 1$, to denote the sum of products where one factor appears exactly j_1 times, another factor (distinct from the previous one!) appears exactly j_2 times, and another factor (distinct from the previous two!) appears exactly j_3 times, etc. This leaves us with exactly k distinct factors each having multiplicity $j_1,j_2,\ldots,j_k$ respectively. This sum appears with

$$\binom{s}{j_1,j_2,\ldots,j_k} = \frac{s!}{j_1!\cdot j_2!\cdot\ldots\cdot j_k!} \tag{8.138}$$

repetitions in the expansion of S_1^s, where $s = j_1 + j_2 + \ldots + j_k$.

For $\underline{s=4}$: when expanding S_1^4, we need to take into account

- *$(4) = S_4$ with $\binom{4}{4} = \frac{4!}{4!} = 1$ repetition.*
- *$(3,1) = S_3\cdot\omega(1) - S_4$ with $\binom{4}{3,1} = \frac{4!}{3!\cdot 1!} = 4$ repetitions.*

- $(2,2) = S_2^2 - S_4$ *with* $\binom{4}{2,2} = \frac{4!}{2!\cdot 2!} = 6$ *repetitions.*
- $(2,1,1) = S_2 \cdot \omega(2) - (3,1) = S_2 \cdot \omega(2) - S_3 \cdot \omega(1) + S_4$ *with* $\binom{4}{2,1,1} = \frac{4!}{2!\cdot 1!\cdot 1!} = 12$ *repetitions.*

Hence we get

$$\omega(4) = \frac{1}{4!}[S_1^4 - S_4 - 4 \cdot (S_3 \cdot \omega(1) - S_4) - 6 \cdot (S_2^2 - S_4) \tag{8.139}$$

$$- 12 \cdot (S_2 \cdot \omega(2) - S_3 \cdot \omega(1) + S_4)] \tag{8.140}$$

$$= \frac{1}{4!}[S_1^4 - 4 \cdot S_3 \cdot \omega(1) + 4 \cdot S_4 - 6 \cdot S_2^2 + 6 \cdot S_4 - 12 \cdot S_2\omega(2) \tag{8.141}$$

$$+ 12 \cdot S_3 \cdot \omega(1) - 12 \cdot S_4 - S_4] \tag{8.142}$$

$$= \frac{1}{4!}\left[S_1^4 - 12 \cdot S_2 \cdot \omega(2) + 8 \cdot S_3 \cdot \omega(1) - 3 \cdot (S_4 + S_2^2)\right] \tag{8.143}$$

$$= \frac{S_1^4}{4!} - \frac{S_2}{2}\omega(2) + \frac{S_3}{3}\omega(1) - \frac{(S_4 + 2 \cdot S_2^2)}{8} \tag{8.144}$$

Notice how, as we progress with higher values of s, we build a recursive formula in $\omega(s')$ with $s' \leq s$.

Intuition. *Note that the coefficient of $\omega(2)$ for $s = 4$, is the same as the coefficient for $\omega(1)$ for $s = 3$, and is the same as the 'constant term' for $s = 2$. Similarly, the coefficient of $\omega(1)$ for $s = 4$ is the same as the 'constant term' for $s = 3$. (By convention here, we don't consider the terms $\frac{S_1^s}{s!}$ to not be part of the 'constant term'.)*

Hence, in the recursive formula for $\omega(s)$, we would expect the coefficient of $\omega(s')$ with $s' < s$ to be equal to the 'constant term' in the formula for $\omega(s - s')$.

Notation. *For all $s > l \geq 0$, let us denote $\delta_{s,l}$ to be the coefficient of the term $\omega(l)$ in the recursive formula for $\omega(s)$. By convention, we denote $\delta_{s,0}$ for the 'constant term' in the recursive formula for $\omega(s)$. Hence for all $s \geq 1$, we have*

$$\omega(s) = \frac{S_1^s}{s!} + \delta_{s,s-1} \cdot \omega(s-1) + \delta_{s,s-2} \cdot \omega(s-2) + \ldots + \delta_{s,1} \cdot \omega(1) + \delta_{s,0} \tag{8.145}$$

Hypothesis. *The hypothesis will thus rewrite as*

$$\delta_{s,l} = \delta_{s-l,0} \tag{8.146}$$

for all $s > l \geq 0$, which will prove by induction on s in the next lemma.

Lemma 10. *Let $s \geq 1$. Then*

$$\omega(s) = \frac{S_1^s}{s!} + \delta_{1,0} \cdot \omega(s-1) + \delta_{2,0} \cdot \omega(s-2) + \ldots + \delta_{s-1,0} \cdot \omega(1) + \delta_{s,0} \tag{8.147}$$

PROOF. Let us prove by induction on s that for all $s > l \geq 0$, we have

$$\delta_{s,l} = \delta_{s-l,0} \tag{8.148}$$

. We already verified the cases $s = 1,2,3,4$ in the previous remark. Thus let us suppose the induction hypothesis is true for s, and consider the mapping

$$\Upsilon : (j_1,j_2,\ldots,j_k) \mapsto (j_1,j_2,\ldots,j_k,1) \tag{8.149}$$

where $j_1 \geq j_2 \geq \ldots \geq j_k \geq 1$ and $s = j_1 + j_2 + \ldots + j_k$, mapping a partition of s onto a partition of $s+1$.

If we suppose that $(j_1,j_2,\ldots,j_k)$ consists of exactly r 1's, then we can write

$$(j_1,j_2,\ldots,j_k) = c_r \cdot \omega(r) + c_{r-1} \cdot \omega(r-1) + \ldots + c_1 \cdot \omega(1) + c_0 \tag{8.150}$$

for some coefficients $c_r,c_{r-1},\ldots,c_1,c_0$, and with

$$\binom{s}{j_1,j_2,\ldots,j_k} = \frac{s!}{j_1! \cdot j_2 \cdot \ldots \cdot j_k!} \tag{8.151}$$

repetitions in the expansion of S_1^s.

The contribution of $(j_1,j_2,\ldots,j_k)$ to the coefficient $\delta_{s,r'}$ of $\omega(r')$ with $r' \leq r < s$, in the final recursive formula of $\omega(s)$ will be

$$\frac{c_{r'}}{j_1! \cdot j_2! \cdot \ldots \cdot j_k!} \tag{8.152}$$

(keeping in mind that we are dividing by $s!$ after having done all the substractions from S_1^s).

Meanwhile,

$$(j_1,j_2,\ldots,j_k,1) = c_r \cdot \omega(r+1) + c_{r-1} \cdot \omega(r) + \ldots + c_1 \cdot \omega(2) + c_0 \cdot \omega(1) + \tilde{c_0} \tag{8.153}$$

for some coefficient $\tilde{c_0}$, with

$$\binom{s+1}{j_1,j_2,\ldots,j_k,1} = \frac{(s+1)!}{j_1! \cdot j_2 \cdot \ldots \cdot j_k!} \tag{8.154}$$

repetitions in the expansion of S_1^{s+1}.

The contribution of $(j_1,j_2,\ldots,j_k,1)$ to the coefficient $\delta_{s+1,r'+1}$ of $\omega(r'+1)$ with $r' \leq r < s$, in the final recursive formula of $\omega(s+1)$ will be

$$\frac{c_{r'}}{j_1! \cdot j_2! \cdot \ldots \cdot j_k!} \tag{8.155}$$

(keeping in mind that we are dividing by $(s+1)!$ after having done all the substractions from S_1^{s+1}).

Conversely, the coefficient $\delta_{s+1,r'+1}$ only receives contributions from partitions of $(s+1)$ having at least $(r'+1)$ 1's, which correspond exactly to the contributions from the partitions

of s having at least r' 1's. Hence

$$\delta_{s+1,r'+1} = \delta_{s,r'} \tag{8.156}$$

Then by the induction hypothesis, we have $\delta_{s,r'} = \delta_{s-r',0}$. In other words

$$\delta_{s+1,r'+1} = \delta_{s-r',0} \tag{8.157}$$

which completes the proof by induction.

□

Remark 12. *Note that all the coefficients $\delta_{s,l}$ consist of linear combination of products with factors equal to S_j with $j \geq 2$, which are known to converge as $T \to \infty$. Thus those can be considered constants when doing an asymptotic analysis in the subsequent sub-subsections. Also note that $\delta_{s,s-1} = \delta_{1,0} = 0$.*

Proposition 4. *For all $s \geq 1$, we have*

$$\omega(s) = \sum_{r=0}^{s} \psi_{s-r} \frac{S_1^r}{r!} \tag{8.158}$$

where for $l \geq 2$

$$\psi_l = \sum_{k=1}^{l-1} \sum_{(j_1,j_2,\ldots,j_k)\in\Psi_{l,k}} \delta_{j_1,0} \cdot \ldots \cdot \delta_{j_k,0} \tag{8.159}$$

with

$$\Psi_{l,k} = \{(j_1,j_2,\ldots,j_k) \text{ with } j_1 \geq \ldots \geq j_k > 1 \text{ and } j_1 + \ldots + j_k = l\} \tag{8.160}$$

and where we define $\psi_0 = 1$ and $\psi_1 = 0$.

PROOF. For $l \geq 2$, we have

$$\psi_l = \sum_{k=1}^{l-1} \sum_{(j_1,j_2,\ldots,j_k)\in\Psi_{l,k}} \delta_{j_1,0} \cdot \ldots \cdot \delta_{j_k,0} \tag{8.161}$$

$$= \delta_{l,0} + \sum_{k=1}^{l-1} \left(\sum_{j=2}^{l-2} \sum_{(j_2,\ldots,j_k)\in\Psi_{l-j,k-1}} \delta_{j,0} \cdot \delta_{j_2,0} \cdot \ldots \cdot \delta_{j_k,0} \right) \tag{8.162}$$

$$= \delta_{l,0} + \sum_{j=2}^{l-2} \left(\sum_{k=1}^{l-j} \sum_{(j_2,\ldots,j_k)\in\Psi_{l-j,k-1}} \delta_{j,0} \cdot \delta_{j_2,0} \cdot \ldots \cdot \delta_{j_k,0} \right) \tag{8.163}$$

$$= \delta_{l,0} + \sum_{j=2}^{l-2} \delta_{j,0} \cdot \left(\sum_{k=1}^{l-j} \sum_{(j_2,\ldots,j_k)\in\Psi_{l-j,k-1}} \delta_{j_2,0} \cdot \ldots \cdot \delta_{j_k,0} \right) \tag{8.164}$$

$$= \delta_{l,0} + \sum_{j=2}^{l-2} \delta_{j,0} \cdot \psi_{l-j} \tag{8.165}$$

$$= \sum_{j=1}^{l} \delta_{j,0} \cdot \psi_{l-j} \tag{8.166}$$

$$\tag{8.167}$$

In other words, we have shown that for all $l \geq 2$,

$$\psi_l = \sum_{j=0}^{l-1} \delta_{l-j} \psi_j \tag{8.168}$$

Let us now prove the proposition by induction on s.

The case $\underline{s = 1}$ is trivial by the definition of ψ_0 and ψ_1.

Let us now assume the formula is true for s, and let us prove it for $s+1$. By the previous lemma 4, we know that

$$\omega(s+1) = \frac{S_1^{s+1}}{(s+1)!} + \sum_{l=1}^{s} \delta_{s+1-l,0} \cdot \omega(l) + \delta_{s+1,0} \tag{8.169}$$

$$= \frac{S_1^{s+1}}{(s+1)!} + \sum_{l=1}^{s} \delta_{s+1-l,0} \cdot \left(\sum_{r=0}^{l} \psi_{l-r} \cdot \frac{S_1^r}{r!} \right) + \delta_{s+1,0} \tag{8.170}$$

$$= \frac{S_1^{s+1}}{(s+1)!} + \sum_{l=1}^{s} \sum_{r=0}^{l} \delta_{s+1-l,0} \cdot \psi_{l-r} \cdot \frac{S_1^r}{r!} + \delta_{s+1,0} \tag{8.171}$$

$$= \frac{S_1^{s+1}}{(s+1)!} + \sum_{l=0}^{s} \sum_{r=0}^{l} \delta_{s+1-l,0} \cdot \psi_{l-r} \cdot \frac{S_1^r}{r!} \tag{8.172}$$

$$= \frac{S_1^{s+1}}{(s+1)!} + \sum_{r=0}^{s} \sum_{l=r}^{s} \delta_{s+1-l,0} \cdot \psi_{l-r} \cdot \frac{S_1^r}{r!} \tag{8.173}$$

$$= \frac{S_1^{s+1}}{(s+1)!} + \sum_{r=0}^{s} \sum_{l'=0}^{s-r} \delta_{s+1-r-l',0} \cdot \psi_{l'} \cdot \frac{S_1^r}{r!} \tag{8.174}$$

$$= \frac{S_1^{s+1}}{(s+1)!} + \sum_{r=0}^{s} \psi_{s+1-r} \cdot \frac{S_1^r}{r!} \tag{8.175}$$

$$= \sum_{r=0}^{s+1} \psi_{s+1-r} \cdot \frac{S_1^r}{r!} \tag{8.176}$$

completing the proof by induction. □

Remark 13. *Hence we have shown that for all $s \geq 1$*

$$\omega(s) = \sum_{r=0}^{s} \psi_{s-r} \frac{S_1^r}{r!} = \frac{S_1^s}{s!} + \sum_{r=0}^{s-2} \psi_{s-r} \frac{S_1^r}{r!} \tag{8.177}$$

or in other words

$$\omega(s) = \frac{(\ln \beta_{t-1,t+k-1})^s}{s!} + \sum_{r=0}^{s-2} \psi_{s-r} \frac{(\ln \beta_{t-1,t+k-1})^r}{r!} \sim \frac{(\ln T)^s}{s!} + \sum_{r=0}^{s-2} \psi_{s-r} \frac{(\ln T)^r}{r!} \tag{8.178}$$

as $t+k=T \to \infty$, which is roughly the polynomial in $\ln T$ of degree s we were anticipating.

8.3.3. Estimating θ.

Remark 14. *Let us now recall the definition for all $s \geq 1$,*

$$\theta_{l:k}^{(t)}(s) = \sum_{l \leq i_1 < \ldots < i_s < k} \left(\frac{1}{t+i_s} + 1_{i_s - i_{s-1}=1} \right) \cdot \left(\frac{1}{t+i_{s-1}} + 1_{i_{s-1} - i_{s-2}=1} \right) \cdot \ldots \tag{8.179}$$

$$\ldots \cdot \left(\frac{1}{t+i_2} + 1_{i_2 - i_1 = 1} \right) \cdot \left(\frac{1}{t+i_1} + 1_{i_1=0} \right) \tag{8.180}$$

which we would like to estimate using $\omega_{l:k}^{(t)}(s)$.

In order to build a first intuition, let us look at how it plays out for small values for s.

Notation. *In this subsection we omit the superscript (t) notation because there is no ambiguity. We will also occasionally do the abuse of notation and assume $\omega_{l:k}(0) = 1$ for all $l < k$.*

For $\underline{s = 1}$, we get

$$\theta_{0:k}(1) = 1 + \omega_{0:k}(1) \tag{8.181}$$

For $\underline{s = 2}$, we get

$$\theta_{0:k}(2) = 1 + \omega_{1:k}(1) + \omega_{0:k-1}(1) + \omega_{0:k}(2) \tag{8.182}$$

In what follows, we will use the following recursive formula quite frequently

$$\theta_{0:k}(s+1) = \theta_{0:k-1}(s) + \sum_{j=s}^{k-1} \frac{1}{t+j} \theta_{0:j}(s) \tag{8.183}$$

Hence for $\underline{s = 3}$, we get

$$\theta_{0:k}(3) = 1 + \omega_{1:k-1}(1) + \omega_{0:k-2}(1) + \omega_{2:k}(1) + \omega_{0:k-1}(2) \tag{8.184}$$

$$+ \omega_{1:k}(2) + \sum_{j=2}^{k-1} \frac{\omega_{0:j-1}(1)}{t+j} + \omega_{0:k}(3) \tag{8.185}$$

Now let us further observe that for all $s \geq 1$ and $0 \leq r \leq l$, we have

$$\omega_{l+r:k+r}(s) \leq \omega_{l:k}(s) \leq \omega_{l-r:k-r}(s) \tag{8.186}$$

This implies that

$$1 + 2 \cdot \omega_{1:k}(1) + \omega_{0:k}(2) \leq \theta_{0:k}(2) \leq 1 + 2 \cdot \omega_{0:k-1}(1) + \omega_{0:k}(2) \tag{8.187}$$

and, similarly,

$$1 + 3 \cdot \omega_{2:k}(1) + 3 \cdot \omega_{1:k}(2) + \omega_{0:k}(3) \leq \theta_{0:k}(3) \leq 1 + 3 \cdot \omega_{0:k-2}(1) + 3 \cdot \omega_{0:k-1}(2) + \omega_{0:k}(3) \tag{8.188}$$

Hypothesis. *We can thus see the binomial coefficients arising, and we would expect that in general, we have*

$$\sum_{r=0}^{s} \binom{s}{r} \cdot \omega_{0:k-s+r}(r) \geq \theta_{0:k}(s) \geq \sum_{r=0}^{s} \binom{s}{r} \cdot \omega_{s-r:k}(r) \tag{8.189}$$

Lemma 11. *For all $k \geq s \geq 1$, we have*

$$\sum_{r=0}^{s} \binom{s}{r} \cdot \omega^{(t)}_{0:k-s+r}(r) \geq \theta^{(t)}_{0:k}(s) \geq \sum_{r=0}^{s} \binom{s}{r} \cdot \omega^{(t)}_{s-r:k}(r) \tag{8.190}$$

Proof. Let us prove this lemma by induction on s. The cases $s = 1,2,3$ have already been treated in the previous remark.

Let us now assume that the claim holds for s, and prove it for $s+1$ using the recursive formula

$$\theta_{0:k}(s+1) = \theta_{0:k-1}(s) + \sum_{j=s}^{k-1} \frac{1}{t+j} \theta_{0:j}(s) \tag{8.191}$$

For the lower bound, using the induction hypothesis, we get

$$\begin{aligned}
\theta_{0:k}(s+1) &\geq \sum_{r=0}^{s} \binom{s}{r} \cdot \omega_{s-r:k-1}(r) + \sum_{j=s}^{k-1} \frac{1}{t+j} \sum_{r=0}^{s} \binom{s}{r} \cdot \omega_{s-r:j}(r) && (8.192)\\
&= \sum_{r=0}^{s} \binom{s}{r} \cdot \omega_{s-r:k-1}(r) + \sum_{r=0}^{s} \binom{s}{r} \cdot \sum_{j=s}^{k-1} \frac{1}{t+j} \cdot \omega_{s-r:j}(r) && (8.193)\\
&= \sum_{r=0}^{s} \binom{s}{r} \cdot \omega_{s-r:k-1}(r) + \sum_{r=0}^{s} \binom{s}{r} \cdot \omega_{s-r:k}(r+1) && (8.194)\\
&= \sum_{r=0}^{s} \binom{s}{r} \cdot \omega_{s-r:k-1}(r) + \sum_{r=1}^{s+1} \binom{s}{r-1} \cdot \omega_{s-r+1:k}(r) && (8.195)\\
&= 1 + \omega_{0:k}(s+1) + \sum_{r=1}^{s} \left[\binom{s}{r} + \binom{s}{r-1} \right] \cdot \omega_{s-r+1:k}(r) && (8.196)\\
&= 1 + \omega_{0:k}(s+1) + \sum_{r=1}^{s} \binom{s+1}{r} \cdot \omega_{s-r+1:k}(r) && (8.197)\\
&= \sum_{r=0}^{s+1} \binom{s+1}{r} \cdot \omega_{s-r+1:k}(r) && (8.198)\\
& && (8.199)
\end{aligned}$$

For the upper bound, using the induction hypothesis, we get

$$\theta_{0:k}(s+1) \leq \sum_{r=0}^{s} \binom{s}{r} \cdot \omega_{0:k-1-(s-r)}(r) + \sum_{j=s}^{k-1} \frac{1}{t+j} \sum_{r=0}^{s} \binom{s}{r} \cdot \omega_{0:j-(s-r)}(r) \tag{8.200}$$

$$= \sum_{r=0}^{s} \binom{s}{r} \cdot \omega_{0:k-1-(s-r)}(r) + \sum_{r=0}^{s} \binom{s}{r} \cdot \sum_{j=s}^{k-1} \frac{1}{t+j} \cdot \omega_{0:j-(s-r)}(r) \tag{8.201}$$

$$\leq \sum_{r=0}^{s} \binom{s}{r} \cdot \omega_{0:k-1-(s-r)}(r) + \sum_{r=0}^{s} \binom{s}{r} \cdot \sum_{j=s}^{k-1} \frac{1}{t+j-(s-r)} \cdot \omega_{0:j-(s-r)}(r) \tag{8.202}$$

$$= \sum_{r=0}^{s} \binom{s}{r} \cdot \omega_{0:k-1-(s-r)}(r) + \sum_{r=0}^{s} \binom{s}{r} \cdot \sum_{j'=r}^{k-1-(s-r)} \frac{1}{t+j'} \cdot \omega_{0:j'}(r) \tag{8.203}$$

$$= \sum_{r=0}^{s} \binom{s}{r} \cdot \omega_{0:k-1-(s-r)}(r) + \sum_{r=0}^{s} \binom{s}{r} \cdot \omega_{0:k-(s-r)}(r+1) \tag{8.204}$$

$$= \sum_{r=0}^{s} \binom{s}{r} \cdot \omega_{0:k-1-(s-r)}(r) + \sum_{r=1}^{s+1} \binom{s}{r-1} \cdot \omega_{0:k-(s+1-r)}(r) \tag{8.205}$$

$$= 1 + \omega_{0:k}(s+1) + \sum_{r=1}^{s} \left[\binom{s}{r} + \binom{s}{r-1} \right] \cdot \omega_{0:k-(s+1-r)}(r) \tag{8.206}$$

$$= 1 + \omega_{0:k}(s+1) + \sum_{r=1}^{s} \binom{s+1}{r} \cdot \omega_{0:k-(s+1-r)}(r) \tag{8.207}$$

$$= \sum_{r=0}^{s+1} \binom{s+1}{r} \cdot \omega_{0:k-(s+1-r)}(r) \tag{8.208}$$

$$\tag{8.209}$$

completing the proof by induction. □

Remark 15. *Let us recall that*

$$\omega_{l:k}(r) = \sum_{q=0}^{r} \psi_{r-q} \frac{(\ln \beta_{t+l-1,t+k-1})^q}{q!} \tag{8.210}$$

Thus the difference between the upper-bound and the lower-bound becomes

$$\sum_{r=0}^{s} \binom{s}{r} \left[\omega_{o:k-(s-r)}(r) - \omega_{s-r:k}(r) \right] = \sum_{r=0}^{s} \binom{s}{r} \cdot \left[\sum_{q=0}^{r} \psi_{r-q} \frac{(\ln \beta_{t-1,t+k-(s-r)-1})^q - (\ln \beta_{t+s-r-1,t+k-1})^q}{q!} \right] \tag{8.211}$$

which converges to zero as $T = t + k \to \infty$.

8.3.4. Putting it all together.

Remark 16. *Now it is time to turn to $\chi^{(t)}_{0:k}(s)$ and finally put it all together, so that we can finally estimate*

$$\frac{ds_{t+k}}{dh_t} = \sum_{s=0}^{k} V^s \cdot \chi^{(t)}_{0:k}(s) \tag{8.212}$$

and get the asymptotic estimate when $T = t + k \to \infty$.

Let us recall that

$$\chi^{(t)}_{0:k}(s) = \frac{1}{t+k} \cdot \theta^{(t)}_{0:k}(s) + \frac{1}{t+k-1} \cdot \theta^{(t)}_{0:k-1}(s-1) + \dots \tag{8.213}$$

$$\dots + \frac{1}{t+k-s+1} \cdot \theta^{(t)}_{0:k-s+1}(1) + \frac{1}{t+k-s} + 1_{k=s} \tag{8.214}$$

Using the abuse of notation $\theta_{l:k}(0) = 1$ for $l < k$, we can rewrite it as follows

$$\chi^{(t)}_{0:k}(s) = 1_{k=s} + \sum_{i=0}^{s} \frac{1}{t+k-i} \cdot \theta_{0:k-i}(s-i) \tag{8.215}$$

The idea is to use the inequality from lemma 11, and get a similar result for $\chi^{(t)}_{0:k}(s)$, then show that the lower and upper bound are no more than $\Theta(1/T)$ apart, thus enabling us to eventually get an asymptotic estimate for $\frac{ds_{t+k}}{dh_t}$.

We are also omitting the superscript (t) notation here because of lack of ambiguity.

Lemma 12. *For all $s \geq 0$ and $k \geq 1$, we have*

$$1_{k=s} + \frac{1}{t+k} \cdot \sum_{r=0}^{s} \binom{s+1}{r+1} \cdot \omega_{s-r:k}(r) \leq \chi_{0:k}(s) \leq 1_{k=s} + \frac{1}{t+k-s} \cdot \sum_{r=0}^{s} \binom{s+1}{r+1} \cdot \omega_{0:k-(s-r)}(r) \tag{8.216}$$

Proof. Using the upper-bound of lemma 11, we get

$$\chi_{0:k}(s) = 1_{k=s} + \sum_{i=0}^{s} \frac{1}{t+k-i} \cdot \theta_{0:k-i}(s-i) \tag{8.217}$$

$$\leq 1_{k=s} + \sum_{i=0}^{s} \frac{1}{t+k-i} \cdot \sum_{r=0}^{s-i} \binom{s-i}{r} \cdot \omega_{0:k-s+r}(r) \tag{8.218}$$

$$\leq 1_{k=s} + \frac{1}{t+k-s} \cdot \sum_{i=0}^{s} \sum_{r=0}^{s-i} \binom{s-i}{r} \cdot \omega_{0:k-s+r}(r) \tag{8.219}$$

$$= 1_{k=s} + \frac{1}{t+k-s} \cdot \sum_{r=0}^{s} \left[\sum_{i=0}^{s-r} \binom{s-i}{r} \right] \cdot \omega_{0:k-s+r}(r) \tag{8.220}$$

$$= 1_{k=s} + \frac{1}{t+k-s} \cdot \sum_{r=0}^{s} \binom{s+1}{r+1} \cdot \omega_{0:k-s+r}(r) \tag{8.221}$$

Similarly, using the lower-bound of lemma 11, we get

$$\chi_{0:k}(s) = 1_{k=s} + \sum_{i=0}^{s} \frac{1}{t+k-i} \cdot \theta_{0:k-i}(s-i) \tag{8.222}$$

$$\geq 1_{k=s} + \sum_{i=0}^{s} \frac{1}{t+k-i} \cdot \sum_{r=0}^{s-i} \binom{s-i}{r} \cdot \omega_{s-r:k}(r) \tag{8.223}$$

$$\geq 1_{k=s} + \frac{1}{t+k} \cdot \sum_{i=0}^{s} \sum_{r=0}^{s-i} \binom{s-i}{r} \cdot \omega_{s-r:k}(r) \tag{8.224}$$

$$= 1_{k=s} + \frac{1}{t+k} \cdot \sum_{r=0}^{s} \left[\sum_{i=0}^{s-r} \binom{s-i}{r} \right] \cdot \omega_{s-r:k}(r) \tag{8.225}$$

$$= 1_{k=s} + \frac{1}{t+k} \cdot \sum_{r=0}^{s} \binom{s+1}{r+1} \cdot \omega_{s-r:k}(r) \tag{8.226}$$

□

Lemma 13. *For all $s \geq 0$, we have*

$$\chi_{0:k}(s) = 1_{k=s} + \frac{1}{t+k} \left[\sum_{r=0}^{s} \binom{s+1}{r+1} \cdot \omega_{s-r:k}(r) \right] + \Theta\left(\frac{1}{t+k} \right) \tag{8.227}$$

for all large enough $k > 1$, and where the implicit constants from the $\Theta(.)$ notation are dependent on s.

Proof. Building on the previous lemma 12, and substracting the lower bound from the upper bound, we get

$$\sum_{r=0}^{s} \binom{s+1}{r+1} \cdot \left[\frac{\omega_{0:k-s+r}(r)}{t+k-s} - \frac{\omega_{s-r:k}(r)}{t+k} \right] = \sum_{r=0}^{s} \sum_{q=0}^{r} \binom{s+1}{r+1} \frac{\psi_{r-q}}{q!} \cdot \left[\frac{(\ln \beta_{t-1,t+k-s+r-1})^q}{t+k-s} - \frac{(\ln \beta_{t+s-r-1,t+k-1})^q}{t+k} \right] \tag{8.228}$$

When assuming that for large k, we have

$$(\ln \beta_{t-1,t+k-s+r-1})^q \approx (\ln \beta_{t+s-r-1,t+k-1})^q \tag{8.229}$$

then

$$\frac{(\ln \beta_{t-1,t+k-s+r-1})^q}{t+k-s} - \frac{(\ln \beta_{t+s-r-1,t+k-1})^q}{t+k} \approx \frac{1}{t+k} \cdot \left[\frac{s}{t+k-s} \cdot (\ln \beta_{t-1,t+k-s+r-1})^q \right] \tag{8.230}$$

$$\leq \frac{1}{t+k} \cdot \left[\frac{s}{t+k-s} \cdot (\ln \beta_{t-1,t+k-s+r-1})^s \right] \tag{8.231}$$

$$\leq \frac{\tau_s}{t+k} \tag{8.232}$$

for some $\tau_s > 0$ depending on s, for all sufficiently large k.

In other words, we have

$$\sum_{r=0}^{s}\binom{s+1}{r+1}\cdot\left[\frac{\omega_{0:k-s+r}(r)}{t+k-s}-\frac{\omega_{s-r:k}(r)}{t+k}\right]\leq\frac{\tilde{\tau}_s}{t+k} \tag{8.233}$$

for for some $\tilde{\tau}_s>0$ depending on s, for all sufficiently large k.

Meanwhile, for all large enough k, we have

$$\sum_{r=0}^{s}\binom{s+1}{r+1}\cdot\left[\frac{\omega_{0:k-s+r}(r)}{t+k-s}-\frac{\omega_{s-r:k}(r)}{t+k}\right]\approx\frac{s}{(t+k)(t+k-s)}\cdot\sum_{r=0}^{s}\sum_{q=0}^{r}\binom{s+1}{r+1}\frac{\psi_{r-q}}{q!}\cdot(\ln\beta_{t-1,t+k-s+r-1})^q \tag{8.234}$$

$$\geq\frac{\tau_s'}{(t+k)^2}\cdot\sum_{r=0}^{s}\sum_{q=0}^{r}\binom{s+1}{r+1}\frac{\psi_{r-q}}{q!}\cdot(\ln\beta_{t-1,t+k-s+r-1})^q \tag{8.235}$$

$$\geq\frac{\tau_s'}{(t+k)^2}\cdot\sum_{r=0}^{s}\sum_{q=0}^{r}\frac{\psi_{r-q}}{q!}\cdot(\ln\beta_{t-1,t+k-s+r-1})^q \tag{8.236}$$

$$=\frac{\tau_s'}{(t+k)^2}\cdot\sum_{q=0}^{s}\sum_{r'=0}^{s-q}\frac{\psi_{r'}}{q!}\cdot(\ln\beta_{t-1,t+k-s+r'+q-1})^q \tag{8.237}$$

$$\approx\frac{\tau_s'}{(t+k)^2}\cdot\sum_{q=0}^{s}\left(\sum_{r'=0}^{s-q}\psi_{r'}\right)\cdot\frac{(\ln(t+k))^q}{q!} \tag{8.238}$$

$$\geq\frac{\tau_s''}{(t+k)^2}\cdot\sum_{q=0}^{s}\frac{(\ln(t+k))^q}{q!} \tag{8.239}$$

$$\approx\frac{\tau_s''\cdot\exp[\ln(t+k)]}{(t+k)^2} \tag{8.240}$$

$$=\frac{\tau_s''}{t+k} \tag{8.241}$$

for some $\tau_s',\tau_s''>0$ depending on s. □

Proposition 5. *If V is a normal matrix with eigenvalues λ_1,λ_2,...,λ_n of modulus smaller than 1, then*

$$\frac{ds_T}{dh_t}=P\Lambda_T P^* \tag{8.242}$$

where P^* is the conjugate transpose of the unitary matrix P (independent of T) and where Λ_T is a diagonal matrix satisfying

$$(\Lambda_T)_{ii}\sim T^{-1}\cdot c+T^{\lambda_i-1}\cdot c' \tag{8.243}$$

for some positive real constants c,c', as $T\to\infty$.

PROOF. Let $V = P\Lambda P^*$ be the Schur decomposition of V, with $\Lambda = \text{diag}(\lambda_1,\lambda_2,\ldots,\lambda_n)$. Note that since we supposed that V is normal, we thus have that the Schur matrix Λ is indeed diagonal and is composed of the eigenvalues on the diagonal.

Based on lemma 13, one can show that there exists a function $g : \mathbb{N} \to \mathbb{R}_0^+$ such that

$$\chi_{0:k}(s) = 1_{k=s} + \frac{1}{t+k}\left[\sum_{r=0}^{s}\binom{s+1}{r+1}\cdot\omega_{s-r:k}(r) + g(s)\right] \tag{8.244}$$

Thus

$$\frac{ds_{t+k}}{dh_t} = \sum_{s=0}^{k} V^s \cdot \chi_{0:k}(s) \tag{8.245}$$

$$= V^k + \frac{1}{t+k}\left[\sum_{s=0}^{k} g(s)\cdot V^s + \sum_{s=0}^{k}\sum_{r=0}^{s}\binom{s+1}{r+1}\cdot\omega_{s-r:k}(r)\cdot V^s\right] \tag{8.246}$$

$$= V^k + \frac{1}{t+k}\left[\sum_{s=0}^{k} g(s)\cdot V^s + \sum_{s=0}^{k}\sum_{r=0}^{s}\sum_{q=0}^{r}\binom{s+1}{r+1}\cdot\psi_{r-q}\frac{(\ln\beta_{t+s-r-1,t+k-1})^q}{q!}\cdot V^s\right] \tag{8.247}$$

$$\tag{8.248}$$

Since the eigenvalues of V are of modulus smaller than 1, we can assume that there exists a constant $d > 0$ (dependent on the choice of eigenvalues of V) such that for all $k > d$ we have $V^k \approx 0$.

Furthermore since $V^m = (P\Lambda P^*)^m = P\Lambda^m P^*$ for all $m \in \mathbb{N}_0$, while keeping in mind that we pick $T = t + k$, we can write

$$\Lambda_T = \frac{1}{T}\left[\sum_{s=0}^{d} g(s)\cdot\Lambda^s + \sum_{s=0}^{d}\sum_{r=0}^{s}\sum_{q=0}^{r}\binom{s+1}{r+1}\cdot\psi_{r-q}\frac{(\ln\beta_{t+s-r-1,T-1})^q}{q!}\cdot\Lambda^s\right] \tag{8.249}$$

$$= \frac{1}{T}\left[\sum_{s=0}^{d} g(s)\cdot\Lambda^s + \sum_{s=0}^{d}\sum_{q=0}^{s}\sum_{r=q}^{s}\binom{s+1}{r+1}\cdot\psi_{r-q}\frac{(\ln\beta_{t+s-r-1,T-1})^q}{q!}\cdot\Lambda^s\right] \tag{8.250}$$

$$= \frac{1}{T}\left[\sum_{s=0}^{d} g(s)\cdot\Lambda^s + \sum_{q=0}^{d}\sum_{s=q}^{d}\sum_{r=q}^{s}\binom{s+1}{r+1}\cdot\psi_{r-q}\frac{(\ln\beta_{t+s-r-1,T-1})^q}{q!}\cdot\Lambda^s\right] \tag{8.251}$$

$$= \frac{1}{T}\left[\sum_{s=0}^{d} g(s)\cdot\Lambda^s + \sum_{q=0}^{d}\sum_{s=q}^{d}\sum_{r=q}^{s}\binom{s+1}{r+1}\cdot\psi_{r-q}\frac{(\Lambda\cdot\ln\beta_{t+s-r-1,T-1})^q}{q!}\cdot\Lambda^{s-q}\right] \tag{8.252}$$

$$= \frac{1}{T}\left[\sum_{s=0}^{d} g(s)\cdot\Lambda^s + \sum_{q=0}^{d}\sum_{s'=0}^{d-q}\sum_{r'=0}^{s'}\binom{s'+q+1}{r'+q+1}\cdot\psi_{r'}\frac{(\Lambda\cdot\ln\beta_{t+s'-r'-1,T-1})^q}{q!}\cdot\Lambda^{s'}\right] \tag{8.253}$$

$$\sim \frac{1}{T}\left[\sum_{s=0}^{d} g(s)\cdot\Lambda^s + \sum_{q=0}^{d}\sum_{s'=0}^{d-q}\sum_{r'=0}^{s'}\binom{s'+q+1}{r'+q+1}\cdot\psi_{r'}\frac{(\Lambda\cdot\ln T)^q}{q!}\cdot\Lambda^{s'}\right] \tag{8.254}$$

$$= \frac{1}{T}\left[\sum_{s=0}^{d} g(s)\cdot\Lambda^s\right] + \frac{1}{T}\left[\sum_{q=0}^{d}\frac{(\Lambda\cdot\ln T)^q}{q!}\cdot\left(\sum_{s'=0}^{d-q}\sum_{r'=0}^{s'}\binom{s'+q+1}{r'+q+1}\cdot\psi_{r'}\cdot\Lambda^{s'}\right)\right] \tag{8.255}$$

$$\approx \frac{1}{T}\left[\sum_{s=0}^{d} g(s)\cdot\Lambda^s\right] + \frac{1}{T}\exp(\Lambda\cdot\ln T)\cdot(c_0 + c_1\cdot\Lambda + \ldots + c_d\cdot\Lambda^d) \tag{8.256}$$

$$\sim \frac{c}{T} + \frac{c'}{T}\exp(\Lambda\cdot\ln T) \tag{8.257}$$

for some positive constants c',c,c_0,c_1,...,c_d.

Hence

$$(\Lambda_T)_{ii} \sim c\cdot T^{-1} + c'\cdot T^{\lambda_i - 1} \tag{8.258}$$

□

Theorem 6. *If V is a normal matrix with eigenvalues of modulus smaller than 1, then*

$$\|\frac{ds_T}{dh_t}\| = \Omega(1/T) \tag{8.259}$$

as $T \to \infty$. (here $\|.\|$ is the Frobenius norm.)

Proof. Let us start off with the observation that

$$T^{-1}\cdot c + T^{\lambda_i - 1}\cdot c' = \Omega\left(T^{-\min(1,1-\mathfrak{Re}(\lambda_i))}\right) \tag{8.260}$$

as $T \to \infty$. And thus, by using proposition 5, we get

$$\|\frac{ds_T}{dh_t}\| = \Omega(T^{-\eta}) \tag{8.261}$$

where

$$\eta = \min_{i=1,\ldots,n} \{\min(1, 1 - \mathfrak{Re}(\lambda_i))\} \leq 1 \tag{8.262}$$

□

Remark 17. *Note that V being normal is not a necessary condition for the generality of the theorem to hold. We simply chose V to be normal in order to make the calculations less cumbersome.*

In case V is non-normal, its Schur matrix Λ becomes triangular instead of diagonal. In fact, if $t_{i,j}$ are the off-diagonal elements of Schur matrix of V (with $i < j$), then

$$\|V\| = \sqrt{Tr(V^*V)} = \sqrt{Tr(\Lambda^*\Lambda)} = \sqrt{\sum_{i=1}^{n} |\lambda_i|^2 + \sum_{i<j} |t_{i,j}|^2} \geq \sqrt{\sum_{i=1}^{n} |\lambda_i|^2} \tag{8.263}$$

Thus every lower bound on $\sqrt{\sum_{i=1}^{n} |\lambda_i|^2}$ induces a lower bound on $\|V\|$, and in particular an asymptotic lower bound on the modulus of one of the eigenvalues of $\frac{ds_T}{dh_t}$ induces an asymptotic lower bound on $\|\frac{ds_T}{dh_t}\|$.

8.4. Sparse relevance case with bounded dependency depth

Remark 18. *Similarly to remark 7, we are going to assume for this subsection:*

- *no non-linearity in the hidden-to-hidden connection: $J_t = V$ for all t.*
- *all assumptions from Remark 3.*
- *κ-sparse attention: for each $t \geq 1$, there are at most $\kappa \leq t$ values for i such that $\alpha_{i,t} \neq 0$. (Let us define $\kappa_t = |\{i \text{ such that } \alpha_{i,t} \neq 0\}|$)*
- *uniform attention across attended states: for all $t \geq 1$, and all $i \leq t$ such that $\alpha_{i,t} \neq 0$, we have $\alpha_{i,t} = 1/\kappa_t \geq 1/\kappa$.*

Remark 19. *Similarly to remark 8, let us recall that*

$$X_{i,t} = \left(h_i - \sum_{j=1}^{t} \alpha_{j,t} h_j\right) \cdot \frac{\partial e_{i,t}}{\partial h_i} \tag{8.264}$$

and that

$$\sum_{i=1}^{t} \alpha_{i,t} Y_{i,t} = \sum_{i=1}^{t} \alpha_{i,t} \cdot h_i \cdot \left(\frac{\partial e_{i,t}}{\partial s_{t-1}} - \sum_{j=1}^{t} \alpha_{j,t} \cdot \frac{\partial e_{j,t}}{\partial s_{t-1}} \right) \tag{8.265}$$

$$= \sum_{i=1}^{t} \alpha_{i,t} \cdot h_i \cdot \frac{\partial e_{i,t}}{\partial s_{t-1}} - \sum_{i=1}^{t} \alpha_{i,t} \left(\sum_{j=1}^{t} \alpha_{j,t} \cdot h_j \right) \cdot \frac{\partial e_{i,t}}{\partial s_{t-1}} \tag{8.266}$$

$$= \sum_{i=1}^{t} \alpha_{i,t} \cdot \left(h_i - \sum_{j=1}^{t} \alpha_{j,t} h_j \right) \cdot \frac{\partial e_{i,t}}{\partial s_{t-1}} \tag{8.267}$$

Thus we can see that both expressions have the common factor $\left(h_i - \sum_{j=1}^{t} \alpha_{j,t} h_j\right)$.

By defining

$$A_t = \{i \text{ such that } \alpha_{i,t} \neq 0\} \tag{8.268}$$

we see that

$$h_i - \sum_{j=1}^{t} \alpha_{j,t} h_j = h_i - \frac{1}{\kappa_t} \sum_{j \in A_t} h_j \tag{8.269}$$

and we are going to assume for the sake of simplicity that

$$h_i \approx \frac{1}{\kappa_t} \sum_{j \in A_t} h_j \tag{8.270}$$

and thus $X_{i,t} \approx 0$ *and* $\sum_{i=1}^{t} \alpha_{i,t} Y_{i,t} \approx 0$.

Recalling the expression from corollary 6 and that $f(h_t,c_t) = h_t + c_t$ *by remark 3, and that* $J_t = V$ *for all* t*, this will give for all* $k' \geq 0$

$$E_{k'}^{(t)} = \left(\frac{1_{t \in A_{t+k'}}}{\kappa_{t+k'}} + 1_{k'=0} \right) \cdot I \tag{8.271}$$

and for all $k \geq j$*, we get*

$$F_{k+1,j}^{(t)} = \left(\frac{1_{t+j+1 \in A_{t+k+1}}}{\kappa_{t+k+1}} + 1_{k=j} \right) \cdot V \tag{8.272}$$

Hence by recalling proposition 2, the main expression of interest becomes

$$\frac{ds_{t+k}}{dh_t} = \sum_{s=0}^{k} \bar{\xi}_{0:k}^{(t)}(s) = \sum_{s=0}^{k} V^s \cdot \chi_{0:k}^{(t)}(s) \tag{8.273}$$

where

$$\chi_{0:k}^{(t)}(s) = \sum_{0 \leq i_1 < \ldots < i_s < k} \left(\frac{1_{t+i_s+1 \in A_{t+k}}}{\kappa_{t+k}} + 1_{k-i_s=1} \right) \cdot \left(\frac{1_{t+i_{s-1}+1 \in A_{t+i_s}}}{\kappa_{t+i_s}} + 1_{i_s - i_{s-1}=1} \right) \cdot \tag{8.274}$$

$$\ldots \cdot \left(\frac{1_{t+i_1+1 \in A_{t+i_2}}}{\kappa_{t+i_2}} + 1_{i_2-i_1=1} \right) \cdot \left(\frac{1_{t \in A_{t+i_1}}}{\kappa_{t+i_1}} + 1_{i_1=0} \right) \tag{8.275}$$

Remark 20. *Let us now have a look at how we could potentially simplify the analysis of* $\chi^{(t)}_{0:k}(s)$.

If we further assume V *to be normal we can write*

$$V = P\Lambda P^* \tag{8.276}$$

where $\Lambda = diag(\lambda_1,\lambda_2,\ldots,\lambda_n)$ *is the diagonal matrix consisting of the eigenvalues of* V*, and* P^* *is the conjugate transpose of* P.

Hence, we can rewrite

$$\frac{ds_{t+k}}{dh_t} = \sum_{s=0}^{k} V^s \cdot \chi^{(t)}_{0:k}(s) = P \cdot \left(\sum_{s=0}^{k} \Lambda^s \cdot \chi^{(t)}_{0:k}(s) \right) \cdot P^* \tag{8.277}$$

We can therefore see that the asymptotic behaviour of $\frac{ds_{t+k}}{dh_t}$ *depends largely on the asymptotic behaviour of the modulus of the complex-valued polynomial*

$$p_{0:k}(\lambda) = \sum_{s=0}^{k} \lambda^s \cdot \chi^{(t)}_{0:k}(s) \tag{8.278}$$

and thus

$$\| \frac{ds_{t+k}}{dh_t} \| = \sqrt{\sum_{i=1}^{n} |p_{0:k}(\lambda_i)|^2} \tag{8.279}$$

where $\|.\|$ *is the Frobenius norm. Hence in order to prove that*

$$\| \frac{ds_{t+k}}{dh_t} \| = \Omega(1/\kappa^d) \tag{8.280}$$

for all large enough k *(note that* k *and* κ *here are two different symbols), it would suffice to show that there exists* $\lambda \in \{\lambda_1,\ldots,\lambda_n\}$ *such that, for all large enough* k*, we have*

$$|p_{0:k}(\lambda)| = \Omega(1/\kappa^d) \tag{8.281}$$

For simplicity we are going to assume that for all t*, we have* $\kappa_t = \kappa$.

Let us further define for all $s \geq 1$,

$$f^{(s)}_{0:k}(i_1,\ldots,i_s) = \left(\frac{1_{t+i_s+1 \in A_{t+k}}}{\kappa_{t+k}} + 1_{k-i_s=1} \right) \cdot \left(\frac{1_{t+i_{s-1}+1 \in A_{t+i_s}}}{\kappa_{t+i_s}} + 1_{i_s-i_{s-1}=1} \right) \cdot \ldots \tag{8.282}$$

$$\ldots \cdot \left(\frac{1_{t+i_1+1 \in A_{t+i_2}}}{\kappa_{t+i_2}} + 1_{i_2-i_1=1} \right) \cdot \left(\frac{1_{t \in A_{t+i_1}}}{\kappa_{t+i_1}} + 1_{i_1=0} \right) \tag{8.283}$$

whenever $(i_1, \ldots, i_n)$ *satisfies* $0 \leq i_1 < i_2 < \ldots < i_s < k$, *and*

$$f^{(s)}_{0:k}(i_1, \ldots, i_s) = 0 \tag{8.284}$$

otherwise.

Theorem 7. *Given the* κ*-sparsity assumption and the dependency depth* d*, we have that if* V *is normal and has one positive real eigenvalue, then*

$$\|\frac{ds_{t+k}}{dh_t}\| = \Omega(1/\kappa^d) \tag{8.285}$$

for all large enough k*.*

Proof. By the hypothesis on the dependency depth d, we know that for each k, there exists $s' \leq d$ and $(i_1, i_2, \ldots, i_{s'})$ such that

$$f^{(s')}_{0:k}(i_1, \ldots, i_{s'}) \geq \left(\frac{1}{\kappa}\right)^{s'+1} \geq \left(\frac{1}{\kappa}\right)^{d+1} \tag{8.286}$$

Hence if λ is real and positive, then for all large enough k, we have

$$|p_{0:k}(\lambda)| = \Omega(1/\kappa^d) \tag{8.287}$$

Let us recall that, since V is normal we can write

$$\|\frac{ds_{t+k}}{dh_t}\| = \sqrt{\sum_{i=1}^{n} |p_{0:k}(\lambda_i)|^2} \tag{8.288}$$

where $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of V.

Hence, if V has at least one positive real eigenvalue then

$$\|\frac{ds_{t+k}}{dh_t}\| = \Omega(1/\kappa^d) \tag{8.289}$$

for all large enough k. □

Remark 21. *As already mentioned, since* κ *and* d *are assumed to be constant, the theorem states that*

$$\|\frac{ds_{t+k}}{dh_t}\| = \Omega(1) \tag{8.290}$$

The dependence on κ *and* d *was simply given in order to get an intuition on how* κ *and* d *are influencing the lower bound, and that* d *has more leverage on the lower bound than* κ*.*

Regarding the normality of V*, the same remark can be made as in remark 17.*

Then note that if V *is a (real)* $n \times n$ *matrix, with* n *odd, then we have at least one real eigenvalue. Thus the restriction of having at least one positive real eigenvalue is not that severe.*

Further, one can show that the theorem holds in a slightly more general setting where one might not have at least one positive real eigenvalue.

Let us consider the case where $\kappa = 1$, $|\lambda| < 1$ *such that we could consider* $\lambda^c \approx 0$ *for some large enough positive integer* c*, and that all states between* T *and* $T-c$ *have dependency depth of exactly* d *(where* $T = t + k$*), then*

$$p_{0:k}(\lambda) = \frac{\lambda^d}{\kappa^d} \cdot (1 + \lambda + \ldots + \lambda^{c-d}) = \frac{\lambda^d}{\kappa^d} \cdot \left(\frac{1 - \lambda^{c-d+1}}{1 - \lambda} \right) \tag{8.291}$$

Hence we can see that if we can show that $\left| \frac{1-\lambda^{c-d+1}}{1-\lambda} \right|$ *is lower bounded asymptotically by a constant, independent of* d *and* κ*, (which it is in this case), then we have*

$$|p_{0:k}(\lambda)| = \Omega(1/\kappa^d) \tag{8.292}$$

We also see that we would like λ *to be sufficiently bounded away from a small set of critical values such as the* $(c-d)$*-th roots of unity.*

In a more general setting, we can rewrite

$$p_{0:k}(\lambda) = \frac{\lambda^d}{\kappa^d} \cdot q_{0:k}(\lambda) \tag{8.293}$$

for some polynomial $q_{0:k}$ *with positive real coefficients, and we would like* λ *to be such that* $|q_{0:k}(\lambda)| = \Omega(1)$ *for all sufficiently large* k*.*

Our hypothesis is that the theorem holds as long as λ *is sufficiently bounded away from a small set of critical values in* $\mathbb{C} \setminus \mathbb{R}^+$*, or in other words, we would need only at least one eigenvalue to satisfy this condition. This set of critical values is a dependent on* κ*,* d *and the overall configuration of the attention weights.*

8.5. Tradeoff analysis between sparsity and gradient propagation

As already discussed in Section 4, the sparsity coefficient κ verifies $\kappa = \nu + \rho \geq |S_t| + |R_t|$ for all time step t, where we denote ν for the size of the short-term buffer, and ρ for the maximal size of the relevant sets R_t. In this section we would like to see how gradient propagation varies when changing sparsity. As already discussed at the end of Section 3 as well as at the end of Section 4, decreasing κ, would increasingly force gradients to backpropagate through the recurrent connections, thus degrading gradient stability. Meanwhile, increasing κ would increase the size of the computational graph. Thus we would like to find the optimal trade-off between sparsity and gradient propagation. This trade-off is clearly task-specific and needs to be determined experimentally. The only way to do so is by either changing ν or changing ρ (or both). Hence we are going to analyze the effects on gradient propagation by separately changing ν and ρ.

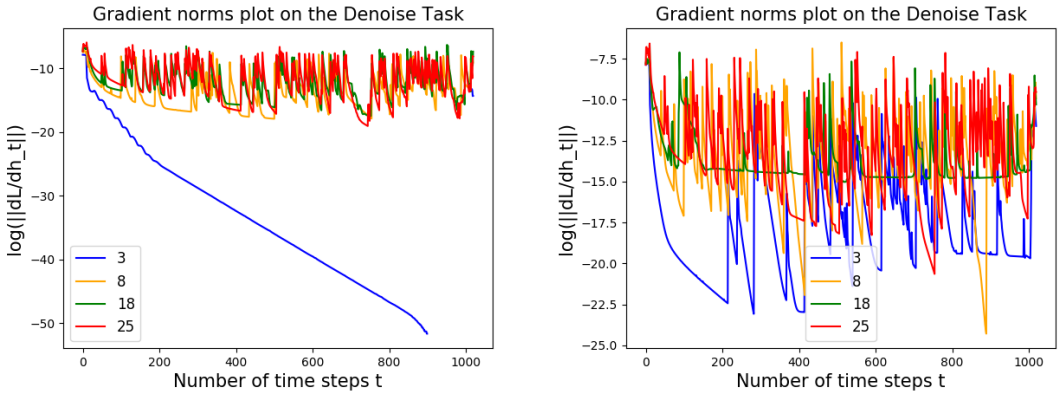

Fig. 14. Both sides show gradient norm plots of $\|\nabla_{h_t} L\|$ in log scale after training for Denoise Task with t ranging from 0 (latest time step) to 1000 (furthest time step). **(Left)** We took four MemLSTM models for $\rho = 3,8,18,25$ while keeping $\nu = 15$ fixed. **(Right)** We took four MemLSTM models for $\nu = 3,8,18,25$ while keeping $\rho = 15$ fixed. (Note that the y-axis of the two plots have different scales, as indicated in the plots.)

T	LSTM	orth-RNN	expRNN	MemRNN	SAB	RelRNN	RelLSTM
100	100%	100%	100%	100%	100%	100%	100%
200	100%	100%	100%	100%	100%	100%	100%
300	100%	100%	100%	100%	100%	100%	100%
500	12%	100%	100%	100%	100%	100%	100%
1000	12%	80%	100%	100%	100%	100%	100%
2000	12%	11%	100%	OOM	100%	100%	100%

Table 8. Results for Copy Task

For Figure 14 (left), we can see that when choosing ρ too small (here for instance $\rho = 3$), gradient propagation becomes unstable, while larger values for ρ all show stable gradient propagation. This confirms our initial intuition that we can decrease ρ until a task-specific treshold and maintain stable gradient propagation, while decreasing ρ beyond this treshold would cause gradient propagation to become unstable.

For Figure 14 (right), we can see that changing ν has much less leverage on gradient propagation than changing ρ. Gradient propagation stays relatively stable regardless of the values for ν. The only difference is that for the extreme value of $\nu = 3$, we can see that gradient propagation became slightly less stable, because with smaller ν predictions for future relevancy might become less accurate.

8.6. Additional Results

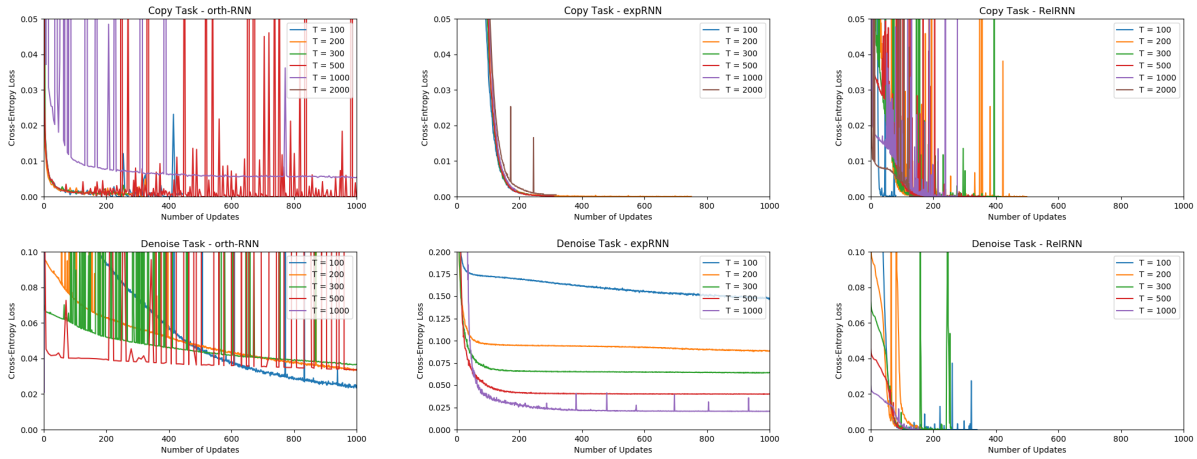

Fig. 15. Cross-entropy vs training updates for Copy (top) and Denoise (bottom) tasks for $T = \{100, 200, 300, 500, 1000, 2000\}$. 1 unit of the x-axis is equal to 100 iterations of training with the exception of expRNN where 1 unit on the x-axis is 10 iterations of training.

Model	lr	optimizer	non-linearity	ν	ρ
orthRNN	0.0002	RMSprop	modrelu	-	-
expRNN	0.0002	RMSprop	modrelu	-	-
LSTM	0.0002	Adam	-	-	-
RelRNN	0.0002	Adam	tanh	10	10

Table 9. Hyperparameters used for Copy task

Model	lr	optimizer	non-linearity	ν	ρ
orthRNN	0.0002	RMSprop	modrelu	-	-
expRNN	0.0002	RMSprop	modrelu	-	-
LSTM	0.0002	Adam	-	-	-
GORU	0.001	RMSprop	-	-	-
RelRNN	0.0002	RMSprop	modrelu	10	10

Table 10. Hyperparameters used for Denoise task

Model	lr (lr orth)	optimizer	non-linearity	ν	ρ
orthRNN	0.0001	Adam	modrelu	-	-
expRNN	0.0001(0.00001)	Adam	modrelu	-	-
LSTM	0.0002		-	-	-
GORU			-	-	
RelRNN	0.0003	Adam	modrelu	10	10

Table 11. Hyperparameters used for sequential MNIST

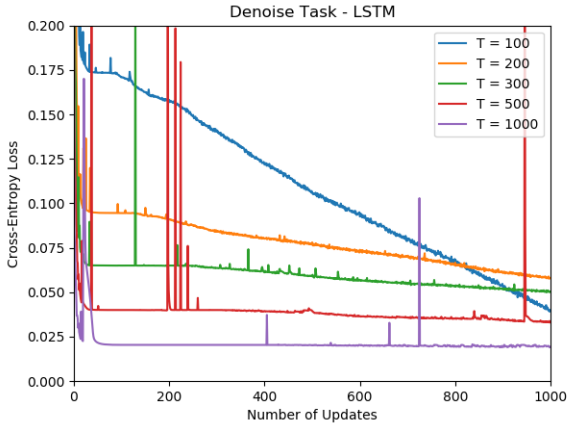

Fig. 16. Training curves for LSTM on Denoise task

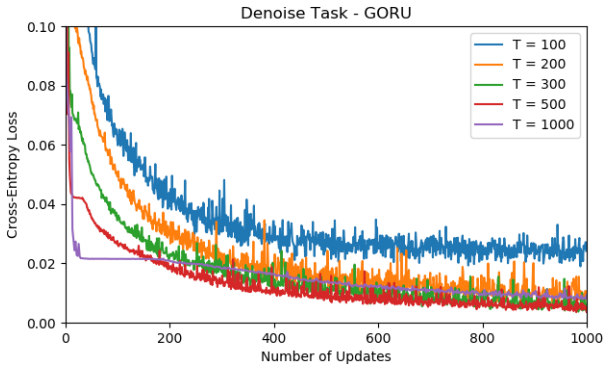

Fig. 17. Training curves for GORU on Denoise task

Model	lr (lr orth)	optimizer	non-linearity	ν	ρ
orthRNN	0.001	Adam	tanh	-	-
expRNN	0.003(0.0003)	Adam	tanh	-	-
LSTM	0.0002		-	-	-
GORU			-	-	
RelRNN	0.0003	Adam	tanh	10	5

Table 12. Hyperparameters used for PTB

Concluding Remarks

In the two articles presented in this thesis we have gained a better understanding of gradient propagation in parametric and semi-parametric, self-attentive RNNs. More precisely, we looked into the components of the gradient that leads to EVGP and prevents the RNN from learning long term dependencies. The first article presented is specific to LSTMs and the proposed algorithm showed significant improvements in its ability to learn tasks of longer sequences and in terms of generalization of the solution learned. The second article presented an algorithm that allows scaling of recurrence and self-attention to larger sequences while, the theorems derived ensure good gradient propagation for the proposed screening mechanism to learn any given task while mitigating EVGP.

The problem of learning long term dependencies in RNN is a difficult one and is yet to be solved. Most proposed solutions only mitigate this problem or solve it for only a particular subset of tasks. Since humans can effortlessly make associations with events across several timescales short and long, solving this problem is an essential step towards reaching the level of human intelligence.

Some of my future work will include testing h-detach on other RNN models like GRUs or self-attentive models. One interesting way to validate the hypothesis presented in the paper would be to explicitly change the backward pass of the algorithm to prevent the suppression of gradients through the cell state. This experiment although not trivial to implement would confirm our hypothesis. Regularizing the recurrent weight matrix norm is also another easy way to to enforce this.

In the second article a possible line of future work is looking into different ways of defining the relevancy screening function in RelLSTM. One could also think of having an additional neural network module that learns to screen for the relevant states. The two articles give rise to several promising directions of future research that could help us take further steps towards solving this problem of long term dependencies in recurrent networks.

References

[1] Cristina Alberini, Sarah Johnson, and Xiaojing Ye. *Memory Reconsolidation*, pages 81–117. 12 2013.

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

[3] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1120–1128. JMLR.org, 2016.

[4] Devansh Arpit, Bhargav Kanuparthi, Giancarlo Kerg, Nan Rosemary Ke, Ioannis Mitliagkas, and Yoshua Bengio. h-detach: Modifying the lstm gradient towards better optimization. *arXiv preprint arXiv:1810.03023*, 2018.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv e-prints*, page arXiv:1409.0473, Sep 2014.

[6] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Trellis networks for sequence modeling. *CoRR*, abs/1810.06682, 2018.

[7] Satanjeev Banerjee and Alon Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[8] Y Bengio, P Simard, and P Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[9] Yoshua Bengio, Paolo Frasconi, and Patrice Simard. The problem of learning long-term dependencies in recurrent networks. In *Neural Networks, 1993., IEEE International Conference on*, pages 1183–1188. IEEE, 1993.

[10] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[11] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. 2016.

[12] Mario Lezcano Casado and David Martínez-Rubio. Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. *CoRR*, abs/1901.08428, 2019.

[13] Maxime Chevalier-Boisvert and Lucas Willems. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

[14] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[15] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[17] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[18] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

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

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

[22] Stephen José Hanson. A stochastic version of the delta rule. *Physica D: Nonlinear Phenomena*, 42(1-3):265–272, 1990.

[23] Anna Harutyunyan, Will Dabney, Thomas Mesnard, Mohammad Gheshlaghi Azar, Bilal Piot, Nicolas Heess, Hado P van Hasselt, Gregory Wayne, Satinder Singh, Doina Precup, and Remi Munos. Hindsight Credit Assignment. pages 12467–12476, 2019.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[25] Mikael Henaff, Arthur Szlam, and Yann LeCun. Recurrent orthogonal networks and long-memory tasks. *arXiv preprint arXiv:1602.06662*, 2016.

[26] S Hochreiter. Untersuchungen zu dynamischen neuronalen netzen [in german] diploma thesis. *TU Münich*, 1991.

[27] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen [in german] diploma thesis. *TU Münich*, 1991.

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

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

[30] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[31] Li Jing, Caglar Gulcehre, John Peurifoy, Yichen Shen, Max Tegmark, Marin Soljacic, and Yoshua Bengio. Gated orthogonal recurrent units: On learning to forget. *Neural computation*, 31(4):765–783, 2019.

[32] Li Jing, Yichen Shen, Tena Dubček, John Peurifoy, Scott Skirlo, Yann LeCun, Max Tegmark, and Marin Soljačić. Tunable efficient unitary neural networks (eunn) and their application to rnns. *arXiv preprint arXiv:1612.05231*, 2016.

[33] Cijo Jose, Moustpaha Cisse, and Francois Fleuret. Kronecker recurrent units. *arXiv preprint arXiv:1705.10142*, 2017.

[34] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3128–3137, 2015.

[35] Nan Rosemary Ke, Anirudh Goyal ALIAS PARTH GOYAL, Olexa Bilaniuk, Jonathan Binas, Michael C Mozer, Chris Pal, and Yoshua Bengio. Sparse attentive backtracking: Temporal credit assignment through reminding. In *Advances in Neural Information Processing Systems*, pages 7651–7662, 2018.

[36] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[37] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[38] David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Hugo Larochelle, Aaron Courville, et al. Zoneout: Regularizing rnns by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305*, 2016.

[39] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

[40] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[41] Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5457–5466, 2018.

[42] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[43] Tsungnan Lin, Bill G Horne, Peter Tino, and C Lee Giles. Learning long-term dependencies in narx recurrent neural networks. *IEEE Transactions on Neural Networks*, 7(6):1329–1338, 1996.

[44] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*, 2017.

[45] Jiasen Lu, Caiming Xiong, Devi Parikh, and Richard Socher. Knowing when to look: Adaptive attention via a visual sentinel for image captioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 6, page 2, 2017.

[46] W. Luo and F. Yu. Recurrent highway networks with grouped auxiliary memory. *IEEE Access*, 7:182037–182049, 2019.

[47] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[48] James Martens and Ilya Sutskever. Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1033–1040. Citeseer, 2011.

[49] Zakaria Mhammedi, Andrew Hellicar, Ashfaqur Rahman, and James Bailey. Efficient orthogonal parametrisation of recurrent neural networks using householder reflections. *arXiv preprint arXiv:1612.00188*, 2016.

[50] Tsendsuren Munkhdalai, Alessandro Sordoni, TONG WANG, and Adam Trischler. Metalearned Neural Memory. pages 13331–13342, 2019.

[51] Ankur P Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inference. *arXiv preprint arXiv:1606.01933*, 2016.

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

[53] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[54] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*, 2017.

[55] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

[56] Gabriel A Radvansky and Jeffrey M Zacks. Event boundaries in memory and cognition. *Current opinion in behavioral sciences*, 17:133–140, 2017.

[57] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[58] Adam Santoro, Ryan Faulkner, David Raposo, Jack Rae, Mike Chrzanowski, Theophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy Lillicrap. Relational recurrent neural networks. In *Advances in Neural Information Processing Systems 31*. 2018.

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

[60] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. 2017.

[61] Dmitriy Serdyuk, Nan Rosemary Ke, Alessandro Sordoni, Adam Trischler, Chris Pal, and Yoshua Bengio. Twin networks: Matching the future for sequence generation. 2018.

[62] Sainbayar Sukhbaatar, arthur szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. In *Advances in Neural Information Processing Systems 28*, pages 2440–2448. 2015.

[63] Mikolov Tomas. Statistical language models based on neural networks. *Brno University of Technology*, 2012.

[64] Trieu H Trinh, Andrew M Dai, Thang Luong, and Quoc V Le. Learning longer-term dependencies in rnns with auxiliary losses. *arXiv preprint arXiv:1803.00144*, 2018.

[65] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 2017.

[66] Ramakrishna Vedantam, C. Lawrence Zitnick, and Devi Parikh. Cider: Consensus-based image description evaluation. *CoRR*, abs/1411.5726, 2014.

[67] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

[68] David Warde-Farley, Ian J Goodfellow, Aaron Courville, and Yoshua Bengio. An empirical analysis of dropout in piecewise linear networks. *arXiv preprint arXiv:1312.6197*, 2013.

[69] Gail Weiss, Yoav Goldberg, and Eran Yahav. On the practical computational power of finite precision rnns for language recognition. *arXiv preprint arXiv:1805.04908*, 2018.

[70] Scott Wisdom, Thomas Powers, John Hershey, Jonathan Le Roux, and Les Atlas. Full-capacity unitary recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 4880–4888, 2016.

[71] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

[72] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5754–5764, 2019.

[73] Ting Yao, Yingwei Pan, Yehao Li, Zhaofan Qiu, and Tao Mei. Boosting image captioning with attributes. In *IEEE International Conference on Computer Vision, ICCV*, pages 22–29, 2017.

[74] Quanzeng You, Hailin Jin, Zhaowen Wang, Chen Fang, and Jiebo Luo. Image captioning with semantic attention. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4651–4659, 2016.

[75] Jeffrey M Zacks, Nicole K Speer, Khena M Swallow, Todd S Braver, and Jeremy R Reynolds. Event perception: a mind-brain perspective. *Psychological bulletin*, 133(2):273, 2007.

[76] Jiong Zhang, Yibo Lin, Zhao Song, and Inderjit S Dhillon. Learning long term dependencies via fourier recurrent units. *arXiv preprint arXiv:1803.06585*, 2018.

[77] Guangxiang Zhao, Junyang Lin, Zhiyuan Zhang, Xuancheng Ren, Qi Su, and Xu Sun. Explicit sparse transformer: Concentrated attention through explicit selection. *arXiv preprint arXiv:1912.11637*, 2019.