institute of neuroinformatics

Universität Zürich[UZH]

ETH Zürich

Continual Learning in Recurrent Neural Networks

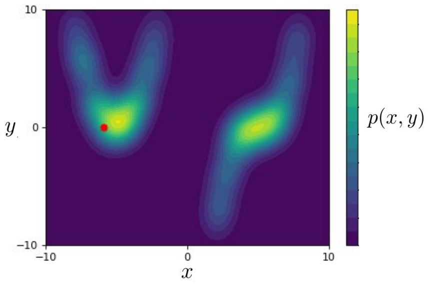
ICLR 2021

Benjamin Ehret*, Christian Henning*, Maria R. Cervera*
Alexander Meulemans, Johannes von Oswald, Benjamin Grewe

Introduction

Continual learning (CL):

How can we learn a set of tasks $\mathcal{D}_1, \ldots, \mathcal{D}_T$ sequentially without obtaining i.i.d. samples from the overall joint $p(x, y)$



CL in RNNs:

Most CL research has been done in feedforward networks, from which **RNNs** differ in two main ways:

- **hidden-to-hidden weights** are sequentially **reused** over time
- **working memory** is needed for solving the tasks

Open questions:

Can existing methods to prevent catastrophic forgetting be used off-the-shelf for RNNs?

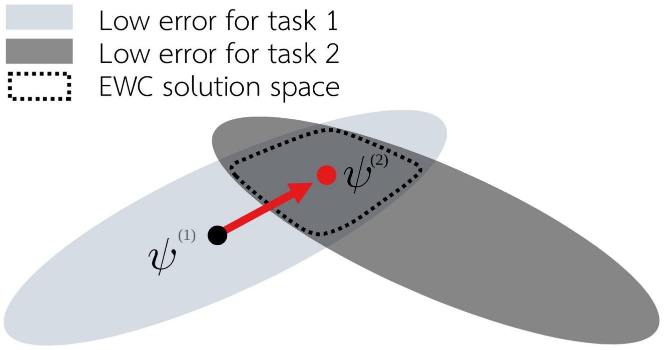
In particular, does any of the two factors above have a particular effect on CL?

Weight-importance methods in RNNs I

Weight-importance methods (e.g. EWC):

Weights are assigned **importance values** for the current task, which affect their **rigidity** for future updates.

[1]

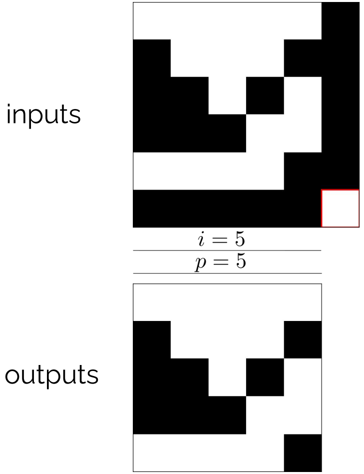


Affected by the **stability-plasticity** dilemma.

[1] Adapted from Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks." PNAS (2017)

Copy task:[2]

Synthetic dataset that allows us to disentangle **working memory requirements** from bare **sequence length**.



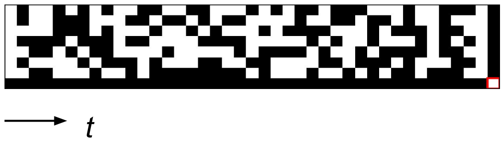
[2] Adapted from Graves, Alex, et al. "Neural Turing Machines." arXiv, 2014

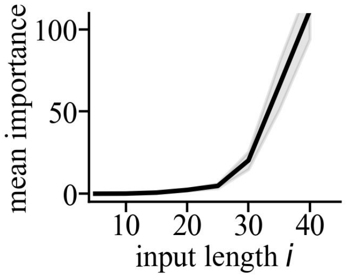
Weight-importance methods in RNNs II

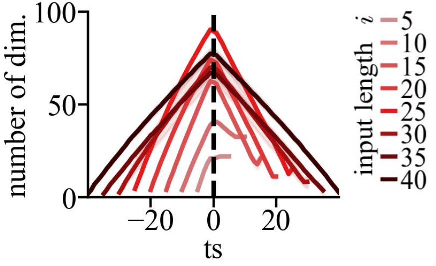
We train networks on **single tasks** with varying pattern and input lengths.

Pattern	Importance values	Intrinsic dimensionality

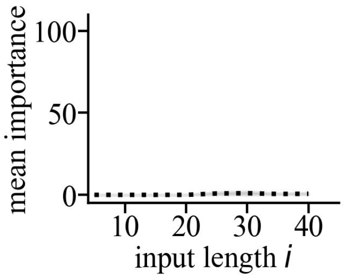
Increased weight reuse **and** working memory (varying *i* and *p*)

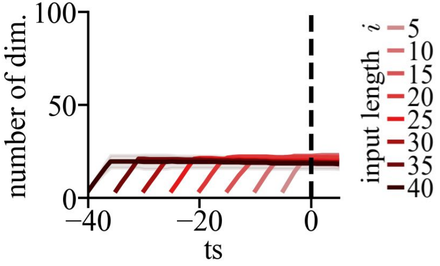






Increased weight reuse (varying *i*, fixed *p*)



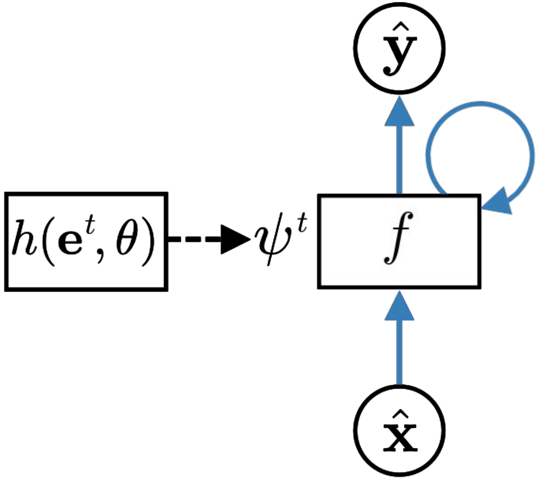


Higher working memory requirements (not weight reuse) lead to increased weight-importance values.

Hypernetworks for CL in RNNs

A hypernetwork *h* **generates the weights** ψ of another neural network *f*.
Task-specific weights can be generated by a single shared hypernetwork.
CL is shifted to the hypernetwork, and forgetting is prevented with a simple regularizer.

hypernetwork *with parameters* θ *and input* ***e***



recurrent

main network *with parameters* ψ *and input* ***x***

The hypernetwork is in theory **agnostic** to the recurrent nature of the task.
Can a feedforward hypernetwork successfully protect a recurrent main model?

Experiments

We systematically compared the performance of different CL methods in RNNs.

We considered a variety of **benchmarks**:

- Audioset
- Copy Task variants
- SplitSMNIST
- Multilingual PoS tagging

And a variety of **methods**:

	during	**final**
Multitask	N/A	77.31 ± 0.10
From-scratch	N/A	79.06 ± 0.11
Fine-tuning	71.95 ± 0.24	49.02 ± 1.00
HNET	73.05 ± 0.45	71.76 ± 0.62
Online EWC	68.82 ± 0.20	65.56 ± 0.35
SI	67.66 ± 0.10	66.92 ± 0.04
...		

We experimentally verified that:

- In weight-importance methods, the stability-plasticity dilemma is aggravated by high working memory requirements
- A CL solution based on hypernetworks can partially overcome this limitation.

Summary & Acknowledgements

- RNNs are affected by catastrophic forgetting in unique ways
- We analysed the use of weight-importance methods for CL in RNNs
  - Working memory requirements directly affect CL
- A systematic comparison of a variety of CL methods in several datasets established that:
  - Despite the mentioned shortcomings, weight-importance methods often remain competitive
  - An approach based on hypernetworks is, however, preferable for CL in RNNs

Thanks to all co-authors

Benjamin Ehret

Christian Henning

Maria R. Cervera

Alexander Meulemans

Johannes von Oswald

Benjamin F. Grewe

Thank you