Maintaining Plasticity in Deep Continual Learning

Rich Sutton and Shibhansh Dohare

Alberta Machine Intelligence Institute
University of Alberta
Reinforcement Learning and Artificial Intelligence Lab

with Fernando Hernandez Garcia, Parash Rahman, and Rupam Mahmood

Main message:

Deep learning does not work for continual learning

- by “not work” i mean that learning slows, eventually to a very low level
- by “deep learning” I mean the standard methods specialized to work in one-time learning
  - without replay buffers (which themselves are an acknowledgement that DL doesn’t work)

Better learning algorithms, specialized for continual learning, are not hard to find.

But we have to start looking for them, we have to get out of the rut!

Earlier indications of problems with deep continual learning

- Catastrophic Forgetting (French, 1999; McCloskey & Cohen, 1989)
- Loss of Plasticity in early neural networks in the psych literature (Ellis & Ralph, 2000; Zevin & Seidenberg, 2002; Bonin et al., 2004)
- The failure of warm-starting (Ash & Adams, 2020)
- Primacy Bias and resetting in Deep RL (Nikishin et al., 2022)
- Capacity Loss in RL (Lyle et al, 2022)

But no one has done a direct test or demonstration of Loss of Plasticity in supervised learning

Plasticity = the ability to learn

Loss of Plasticity = Loss of the ability to learn
= not being able to learn continually
= not continual learning

Maintaining Plasticity = maintaining the ability to learn

At CoLLAs, we prioritize maintaining plasticity

Outline

- Demonstrations of Loss of Plasticity
  (in continual versions of ImageNet, MNIST, and generic regression)
- Understanding Loss of Plasticity
- Existing Methods that try to Maintain Plasticity
- A Simple Extension of Backprop, *Continual Backprop*,
  that Fully Succeeds in Maintaining Plasticity

How can we make a direct test of continual learning?

- We could use a single *non-stationary* task
- or a sequence of different tasks
  with no indication to the learner of the changes
- It could be a reinforcement learning task, but…that’s complicated
- It could be supervised. It could be classic classification. Why not?
- Perhaps we could use a classic deep-learning dataset
  like ImageNet or MNIST?

ImageNet

a classic deep learning problem

ImageNet Dataset

- A database of millions of images labelled by nouns (classes)
- 1000 classes with 700 or more images
- Widely used in to learn classification: image⇒class

⇒ The Continual ImageNet Problem

- Here we seek a minimal change from Deep Learning practice
- Each class separated into 600 training examples and 100 test examples
- Classes taken in pairs to produce a sequence of 500+ binary classification tasks
  - e.g., Class1 vs Class2, 1200 training examples, 200 test examples, then Class3 vs Class4, 1200 training examples, 200 test examples, etc
- Performance measure: %correct on test set (by argmax) at end of each task
- Averaged over 30 independent runs, varying class pairings, test sets

Network and Training Procedure (ImageNet)

- All 500 binary classification tasks share the same network, heads reset at task switch
- Standard neural network, though slightly narrow for ImageNet (bc. only 2 classes at a time)
  (3 convolution layers of 32/64/128 filters + 3 fully-interconnected layers of 128/128/2 units)
- For each task, 12 batches of 100 examples, 250 epochs (passes through the data)
- Weights initialized by Kaiming distribution, only once, before the first task
- SGD with momentum on the cross-entropy loss, ReLU activations
- Many variations on the network and hyper-parameters were tested
  to obtain good and representative performance on the first task

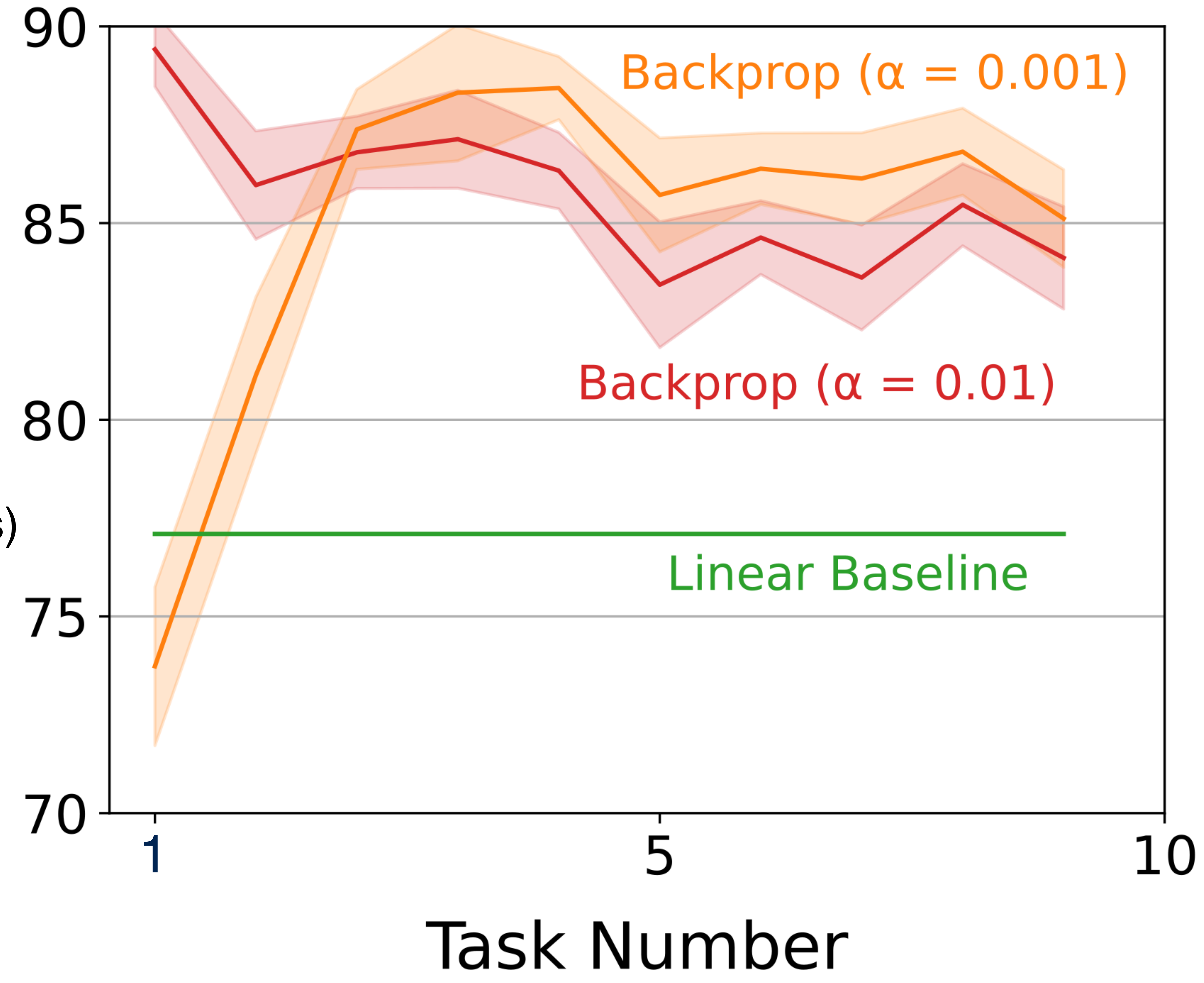
How will performance evolve over the sequence of tasks?

Will performance be better on the 1st task or the 2nd task? the 500th?

Continual ImageNet Results (initial performance)

%Correct on Test Set

(at end of each task, averaged over 30 runs)



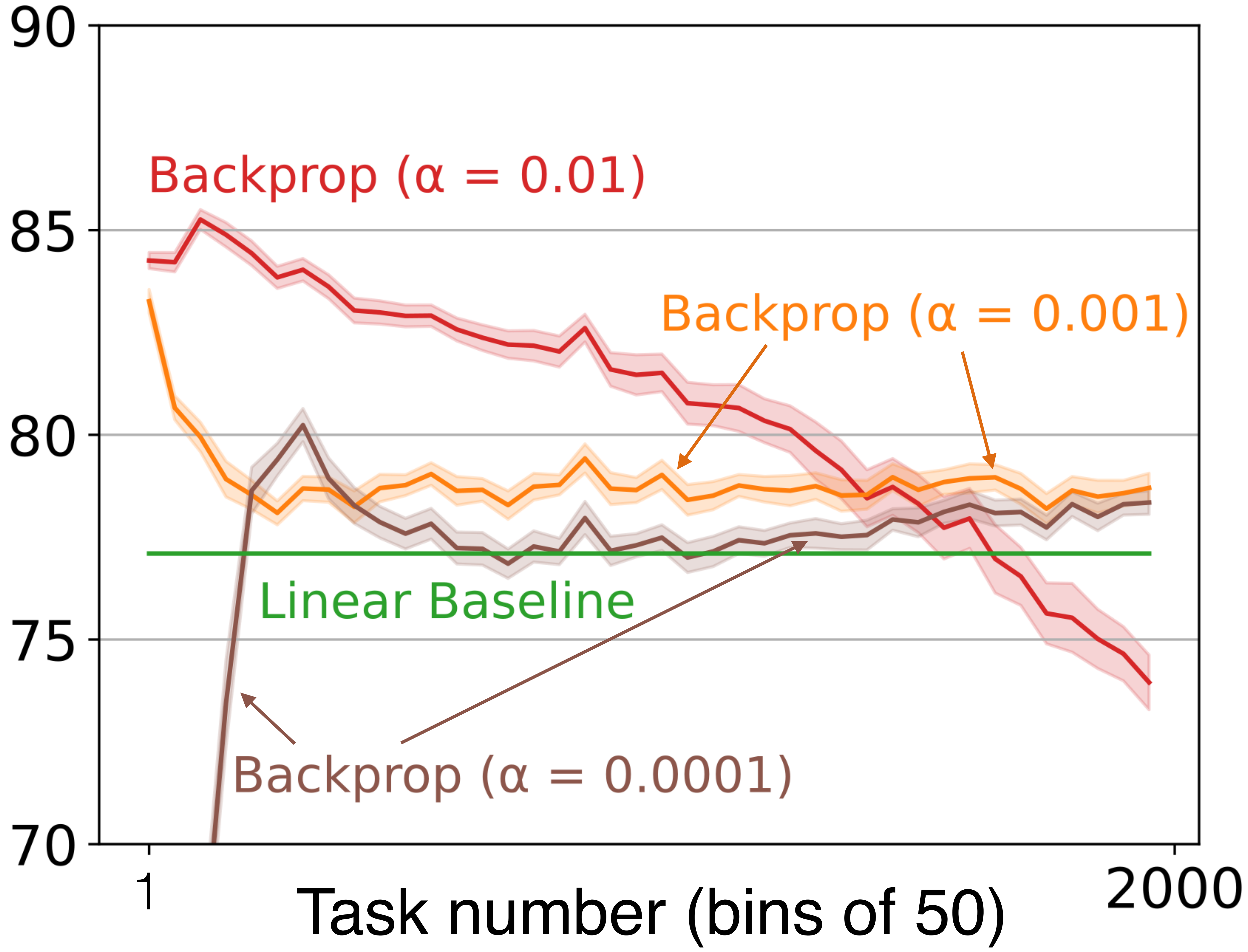
- Chance performance is 50%
- Shaded region is one standard error
- Linear baseline is the performance of linear heads direct from pixels

Learning rate (plasticity) sometimes improves over early tasks, then…?

Continual ImageNet Results — Overview

%Correct
on Test Set

(at end of each task,
averaged over 30 runs)



- Performance on first task is ≈89%
- This data is representative, the details depend on the details:
    - #epochs
    - step-sizes
    - network sizes
- Each line takes 24 hours to compute

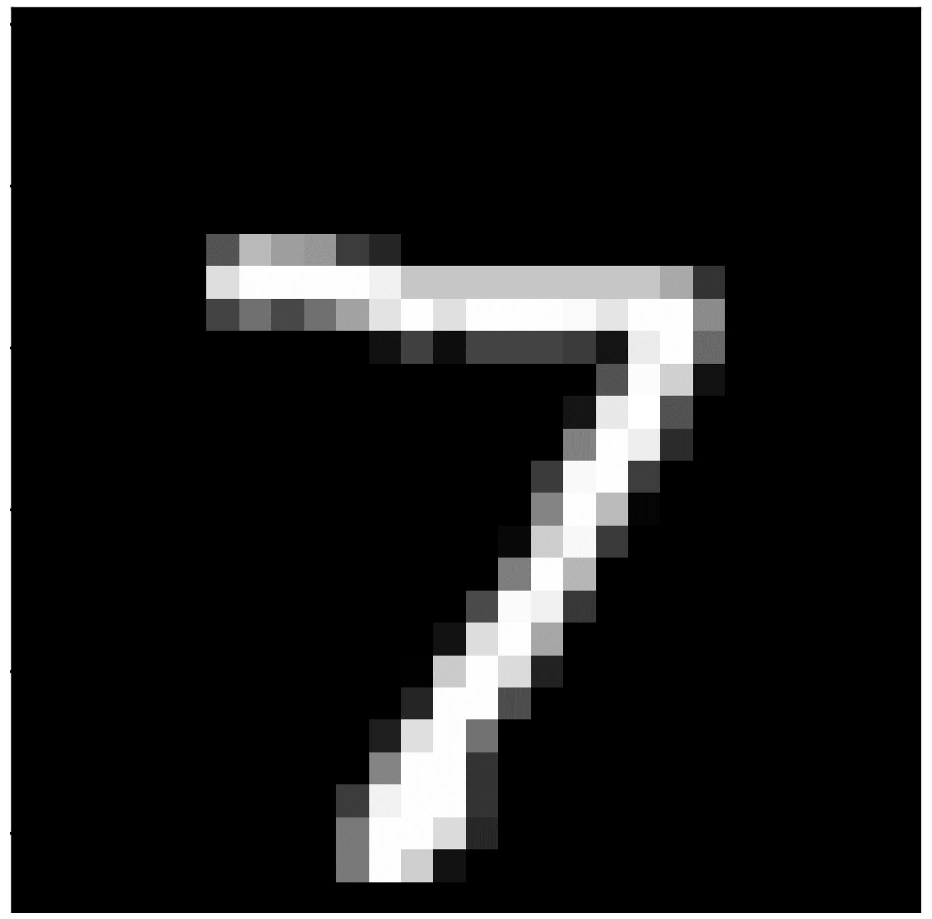
For good hyper-parameters, plasticity decreases across tasks,
nearing the poor performance level of a one-layer (linear) network, or worse

“Catastrophic” Loss of Plasticity

MNIST

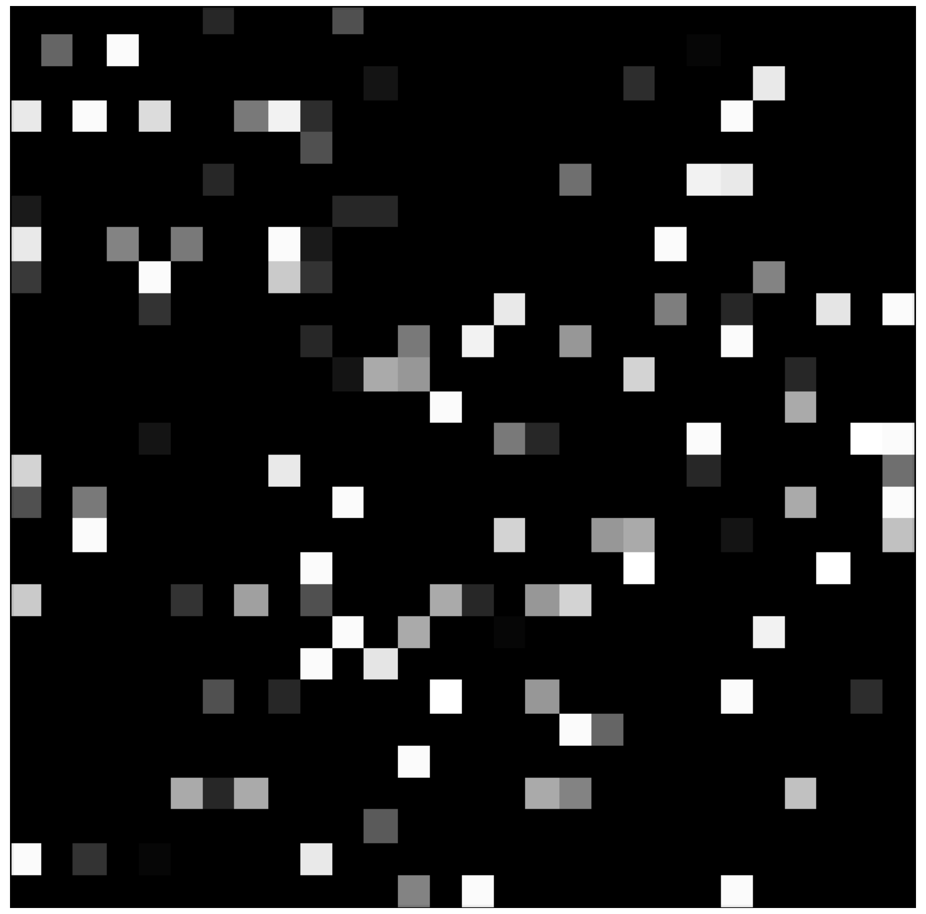
(to run more thorough, systematic experiments)

MNIST dataset



- 60,000 images of handwritten digits, ten classes 0-9, grayscale 28x28 pixels,

Permuted MNIST Task (Goodfellow et al. 2014; Zenke, Poole, & Ganguli 2017)



- The same 60,000 images with the pixels randomly permuted

⇒ The Continual MNIST Problem

- A sequence of Permuted MNIST tasks
- In each task, all 60,000 images presented in a random order
- No indication of the new task (weights initialized once before 1st task)
- Online cross-entropy loss, report argmax %correct

Network and Training Procedure (MNIST)

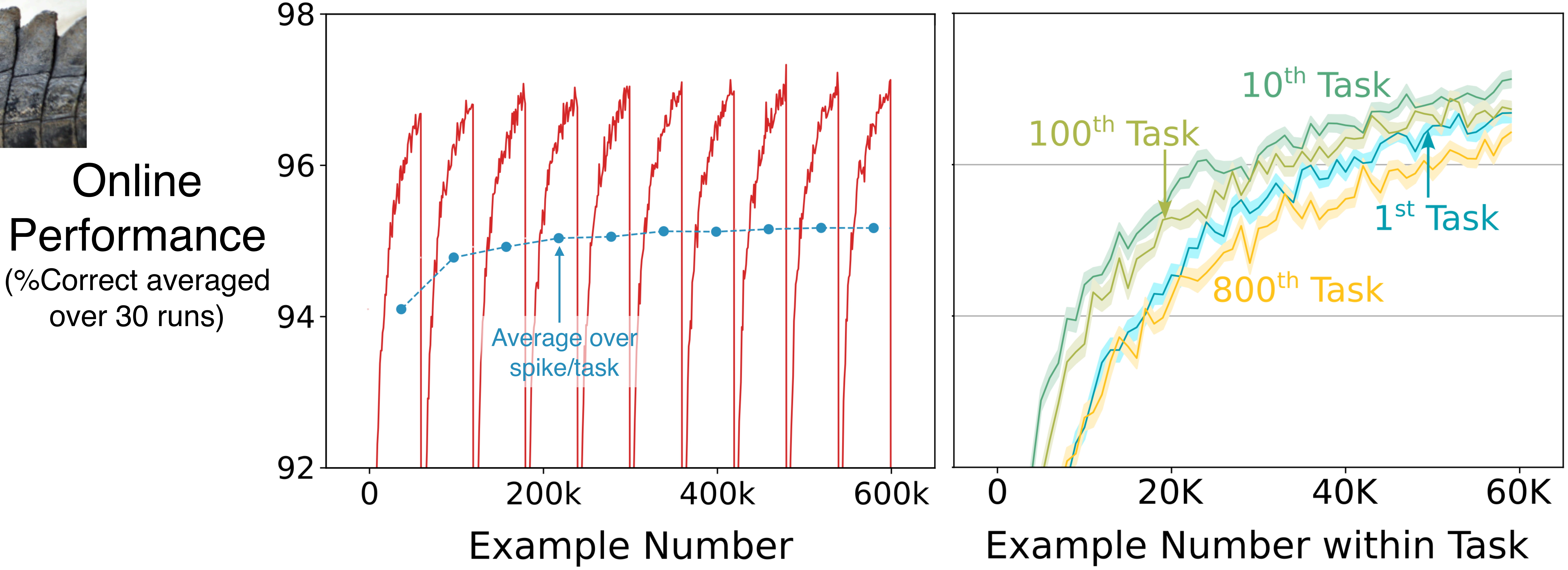
- Standard neural network — 4 fully-interconnected layers of 2000/2000/2000/10 units)
- All one-of-ten classification tasks share the same network
- The 10 heads are not re-initialized at task change
- Weights initialized by Kaiming distribution, only once, before the first task
- SGD on the cross-entropy loss, ReLU activations

Backprop on Continual Permuted MNIST (detail)

Alligator's tail

Alligator graph

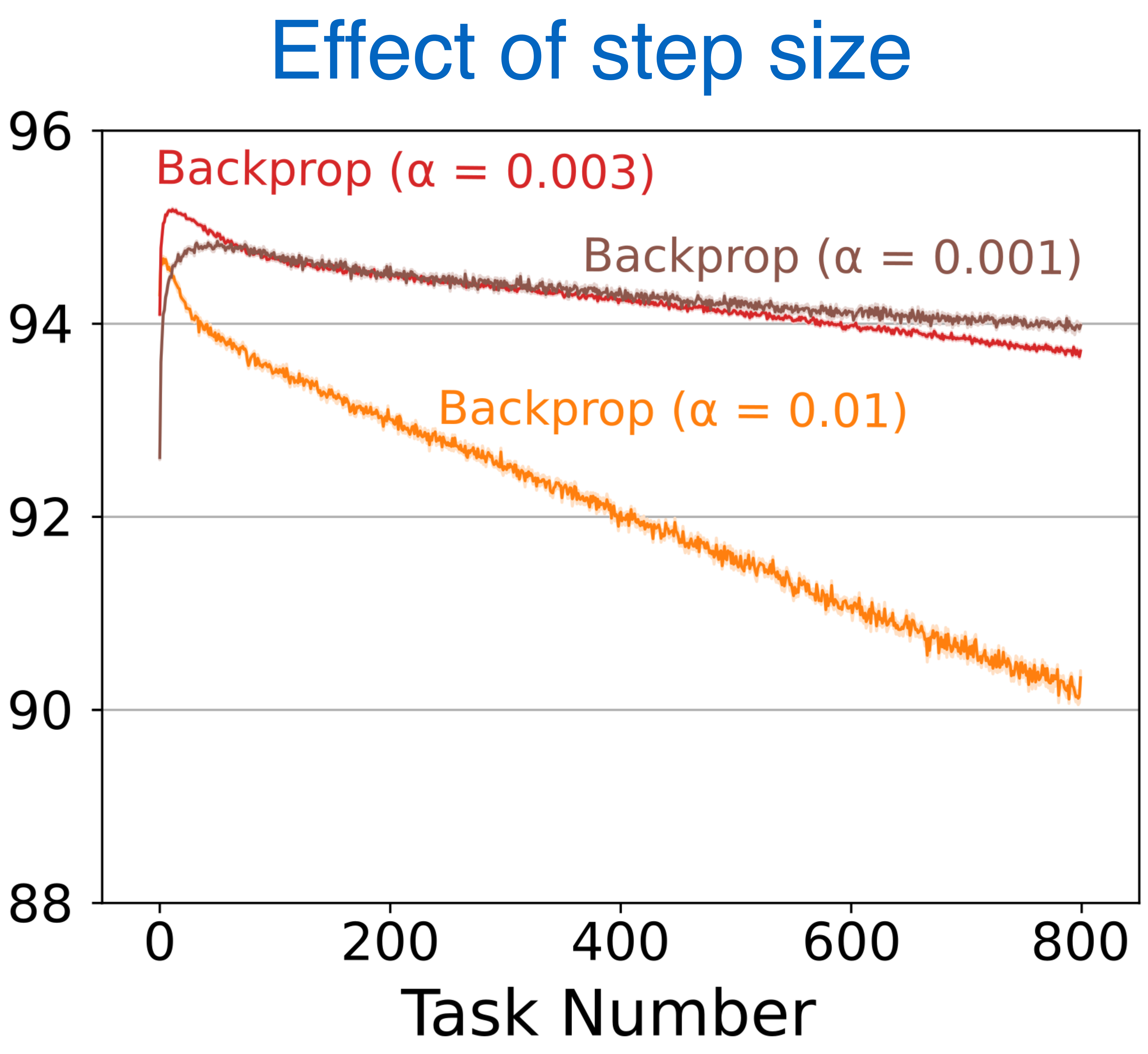
Superimposed spikes

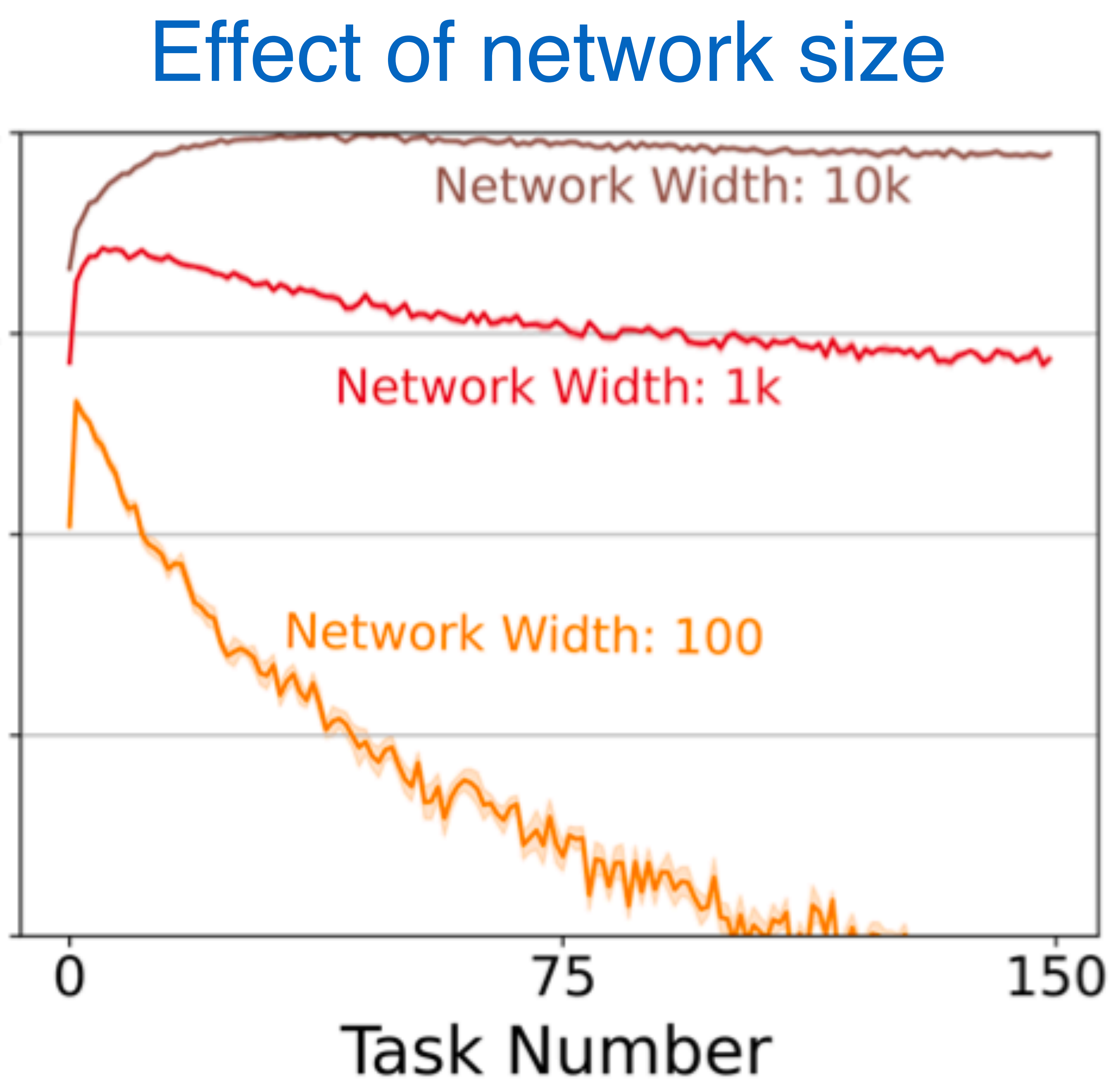


Learning rate (plasticity) improves over early tasks, then degrades

Backprop on Continual Permuted MNIST (overview)

Online Performance
(%Correct averaged over 30 runs)

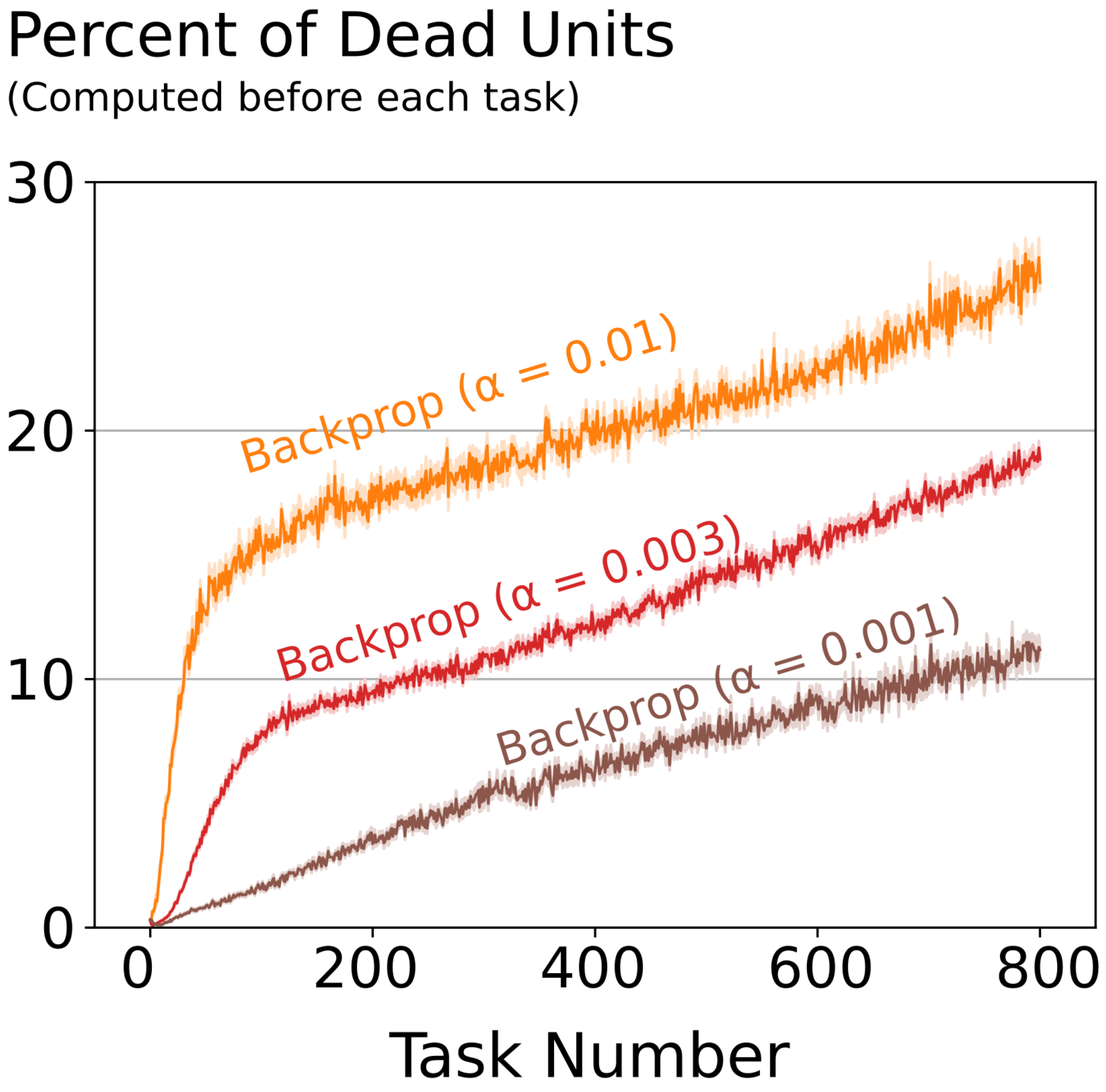


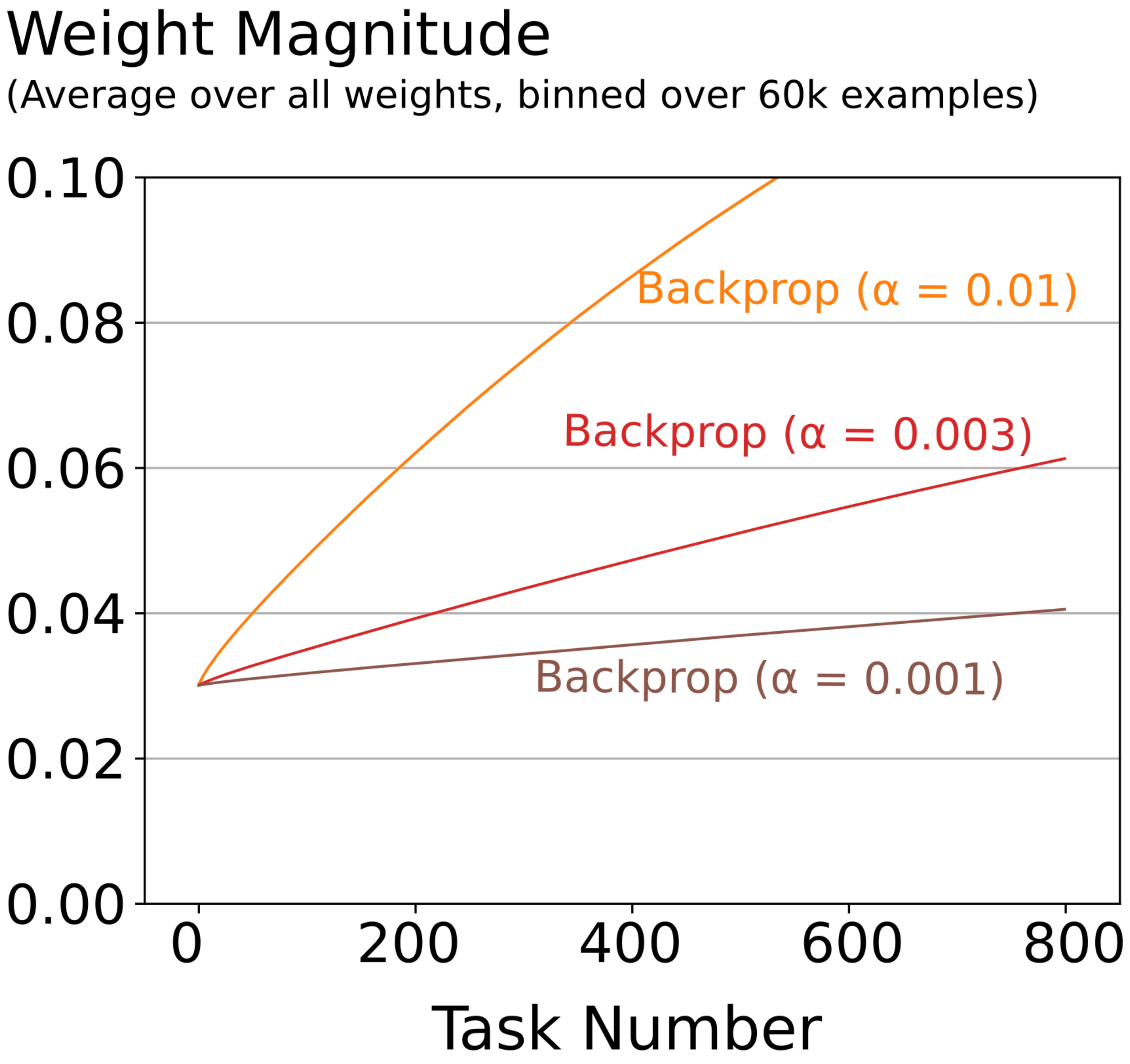


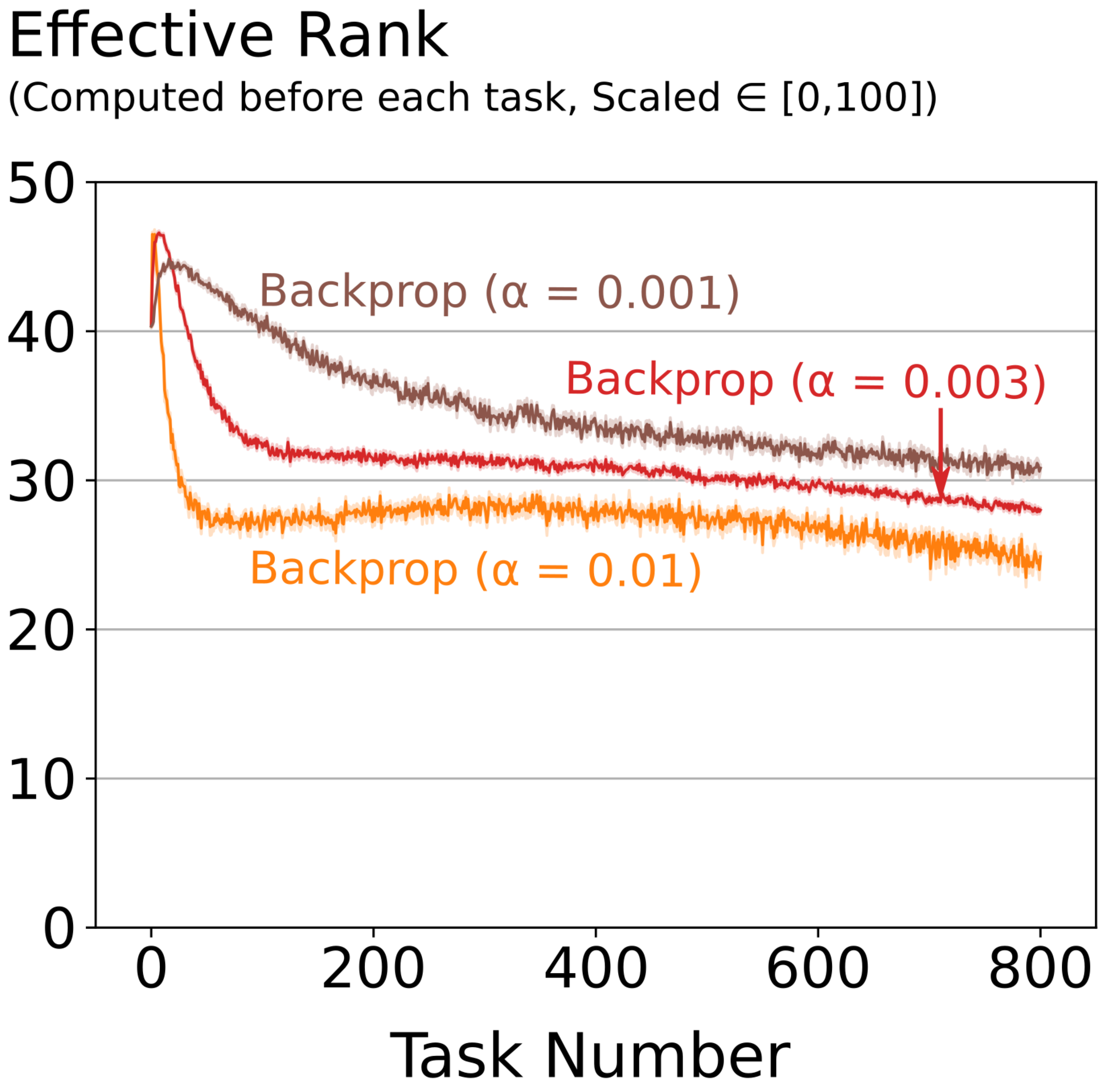
There is substantial loss of plasticity at all step sizes.

Larger networks lose plasticity more slowly, but still lose plasticity.

Understanding loss of plasticity (MNIST)







Units are dying

A *dead unit*'s output (and thus its plasticity) is always zero

Weights are getting large

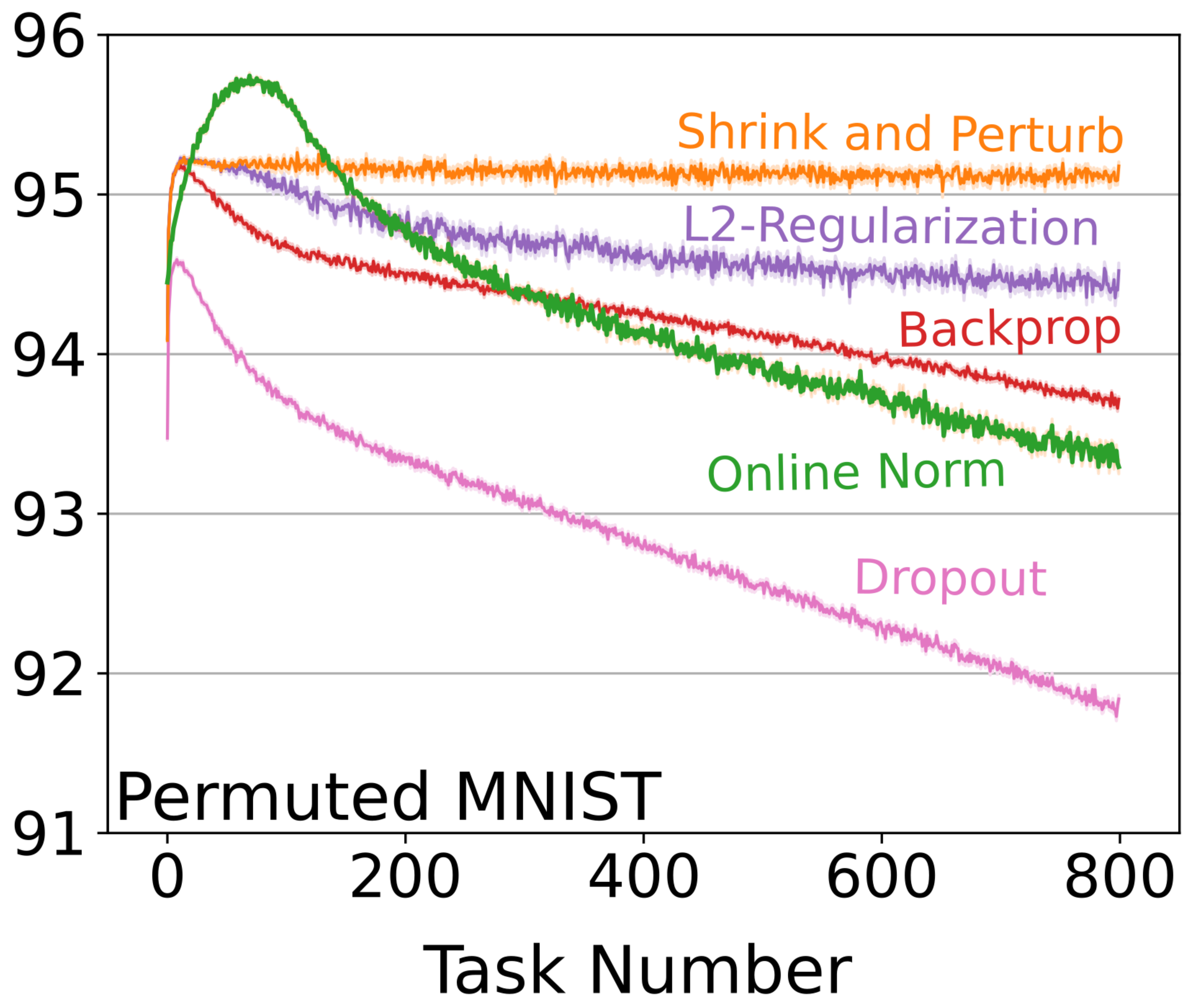
Units are over-committed and have become difficult to change

The effective rank of activity in the representation layers is falling

The representation layers are losing diversity and expressiveness

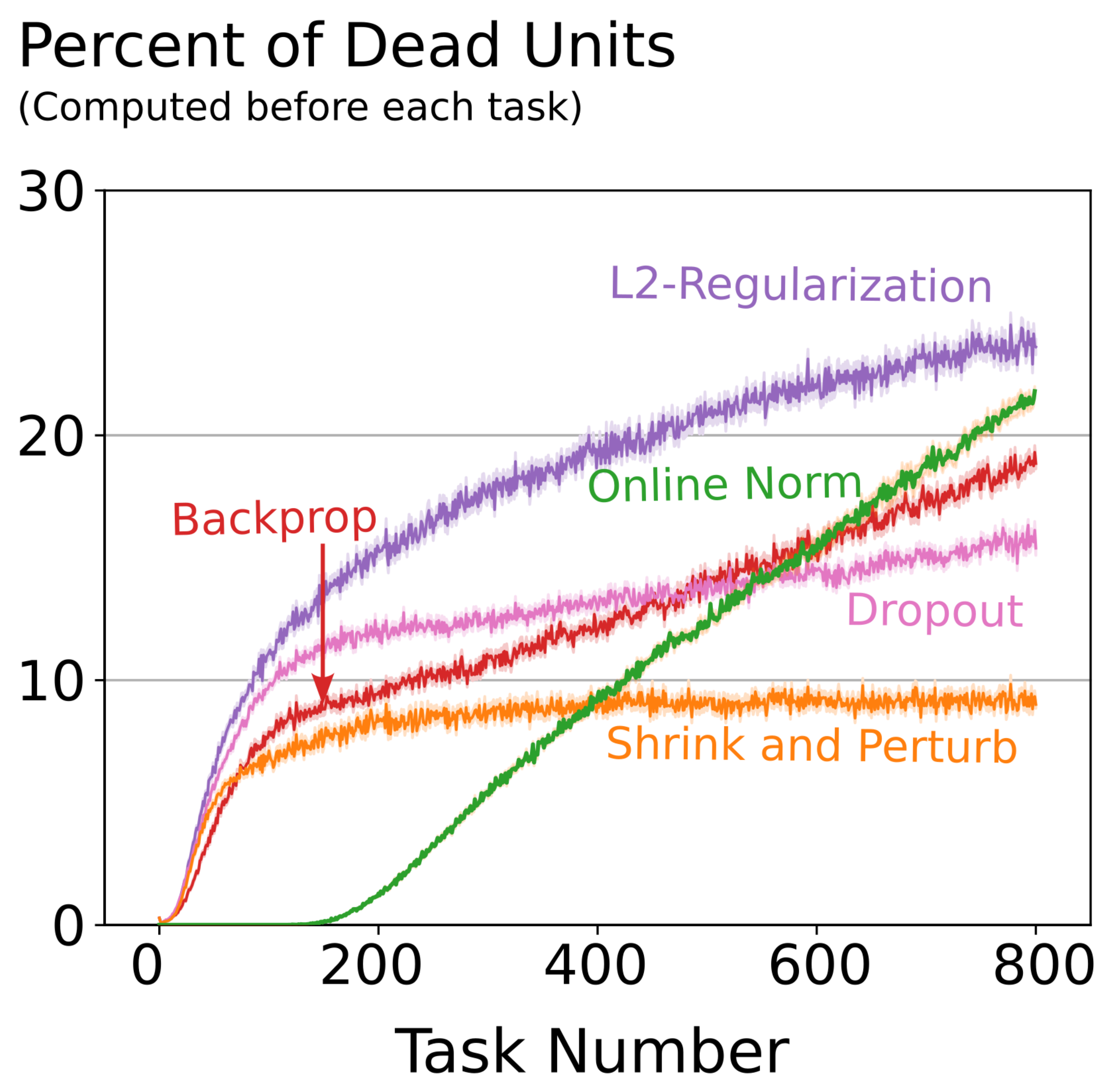
Do existing DL methods help to maintain plasticity?

Online Performance (%Correct averaged over 30 runs)

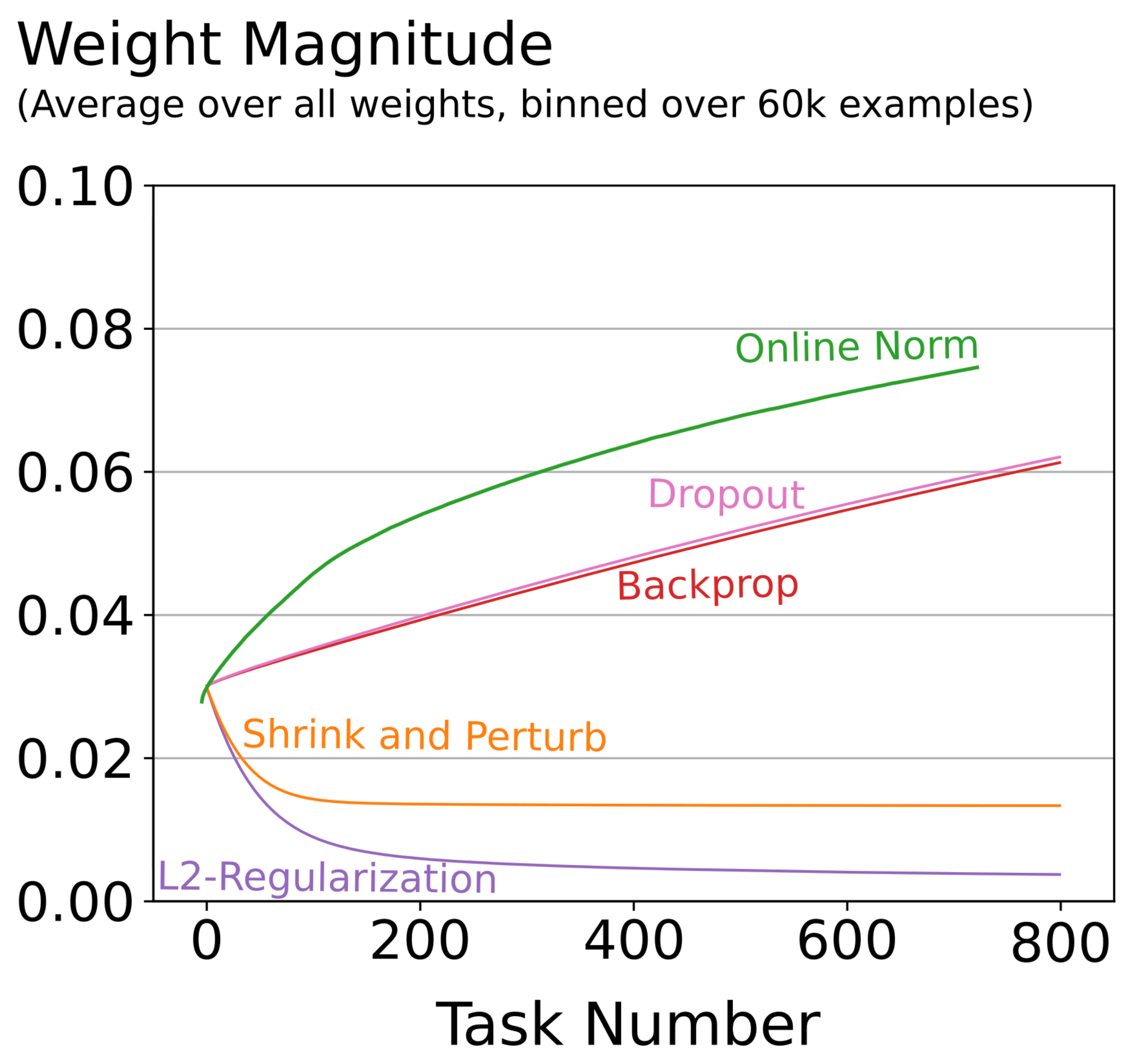


- L2-Regularization: All weights shrunk towards zero on each training example
- Shink and Perturb: L2 regularization plus random noise added to all weights (Ash & Adams 2020)
- Online Normalization: All signals internal to the network are shifted and scaled online (Chiley et al. 2019)
- Dropout: A random fraction of units set to zero in training (Hinton et al. 2012)

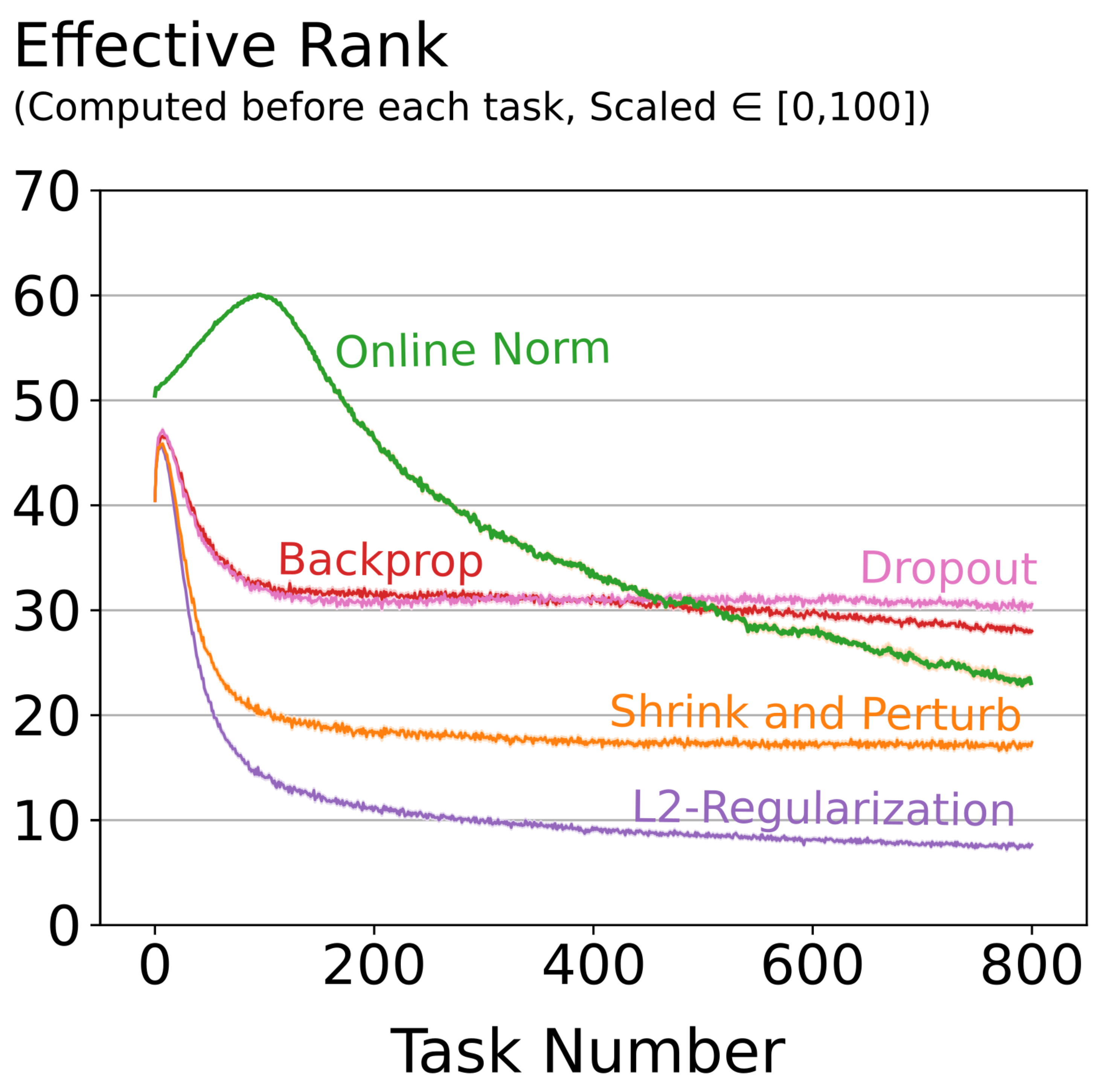
Understanding LoP in Existing DL Methods (MNIST)



Shrink & Perturb (but not L2) keeps units from dying



L2 and Shrinking keep the weights from getting large



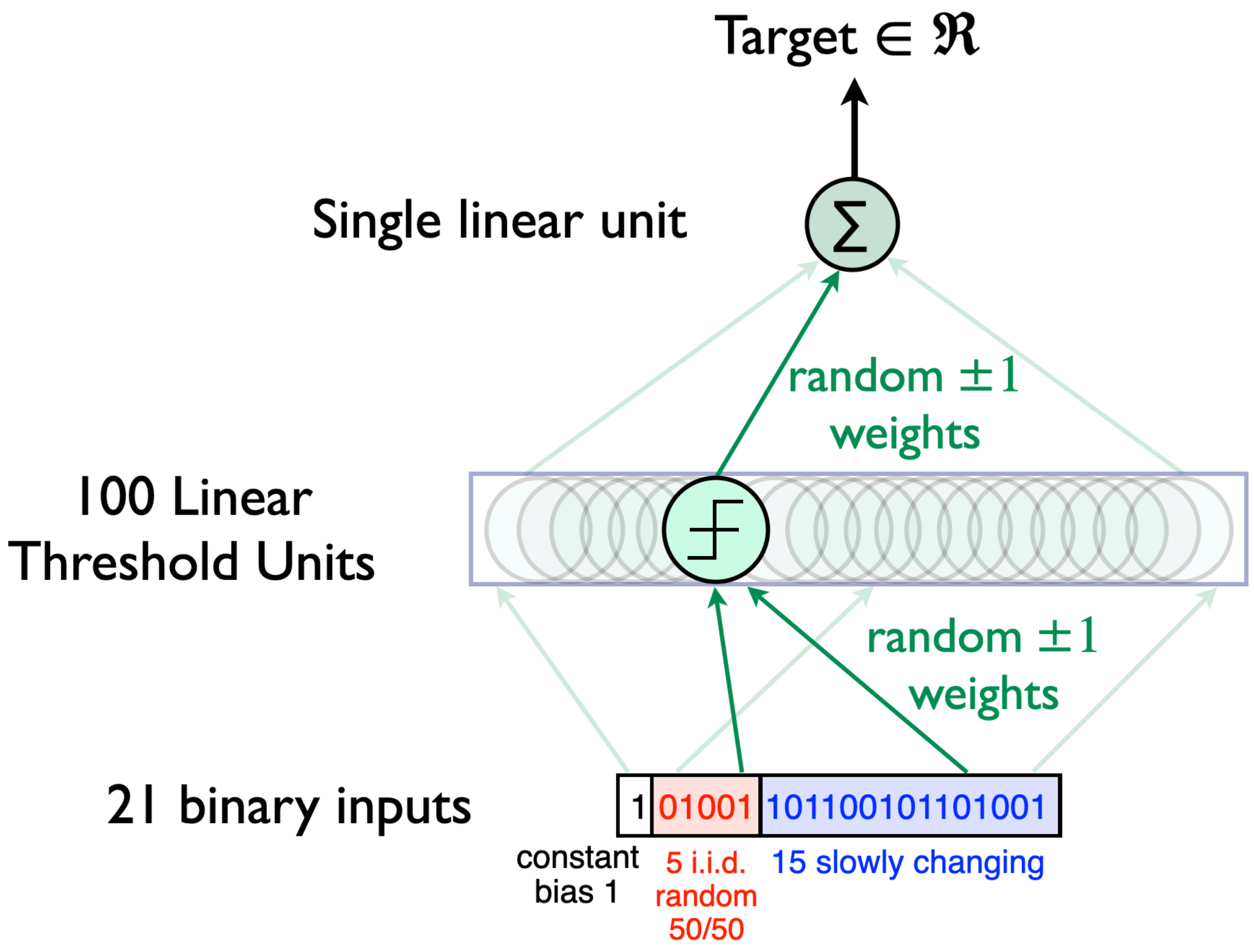
Effective rank is still a problem for L2 and Shrink & Perturb

All our ideas and algorithms came from a still smaller problem:

Slowly-Changing Regression (SCR)

Slowly-Changing Regression (SCR)

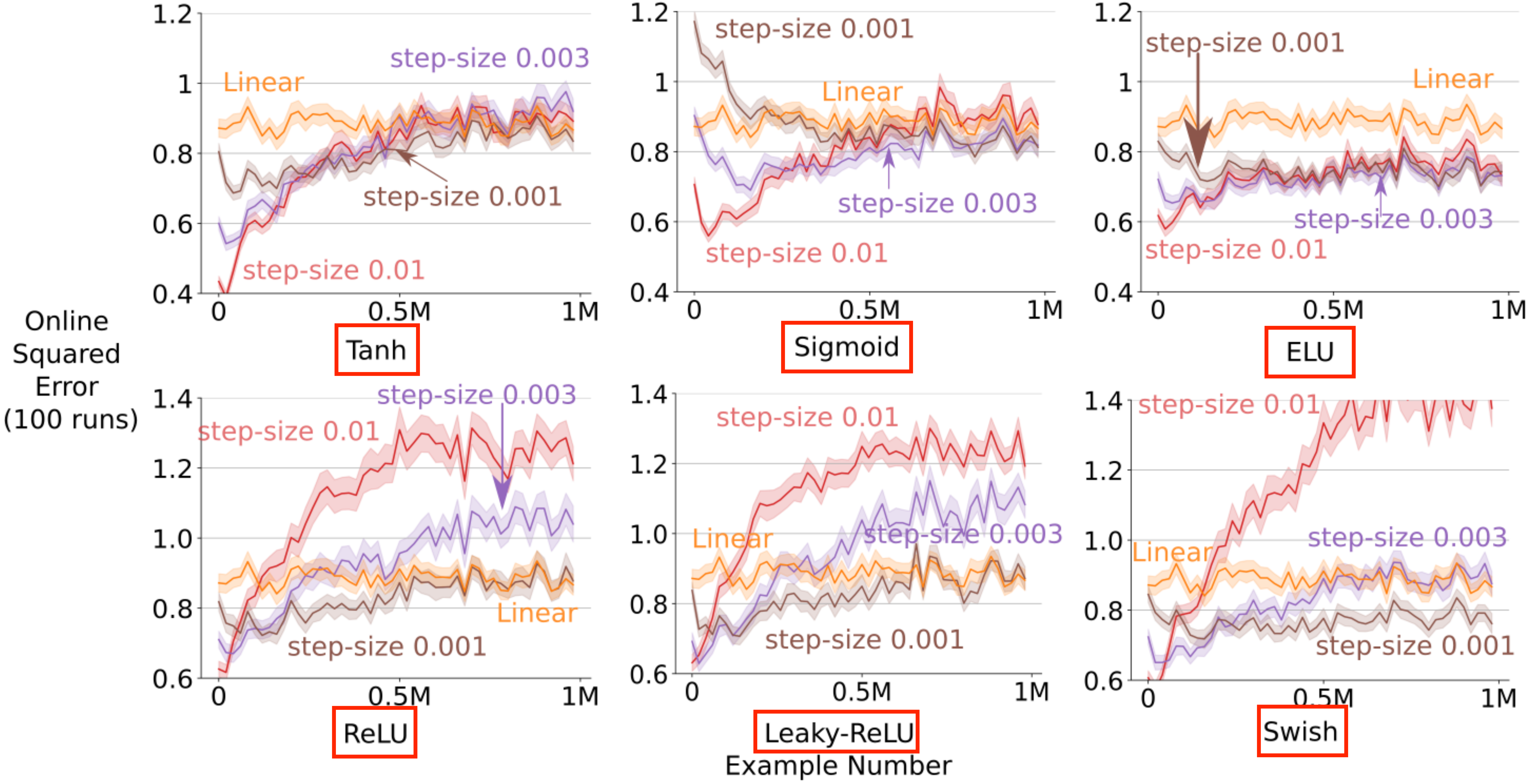
a new idealized problem targeting continual learning



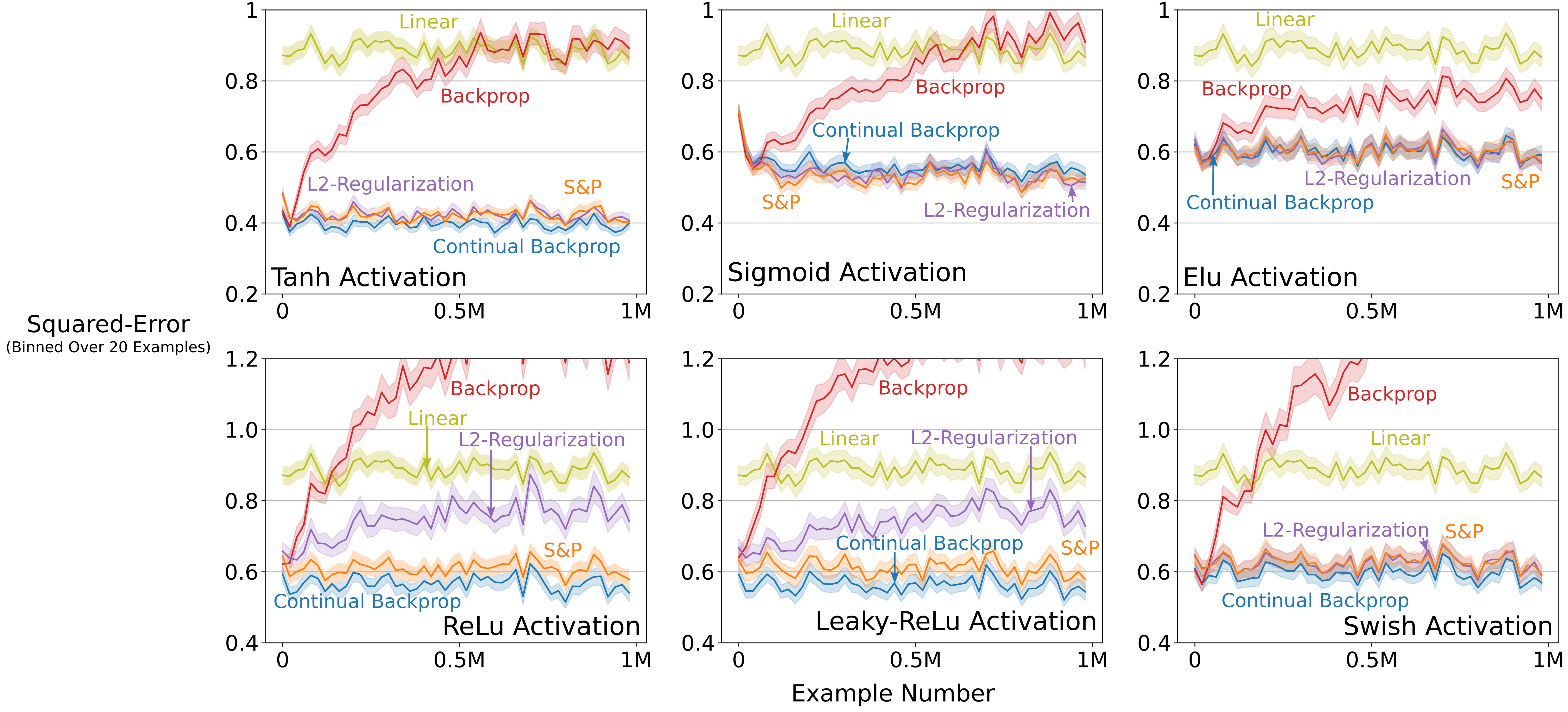
- Target function is that formed by a 1-hidden-layer neural network with random ±1 weights and binary (linear threshold) hidden units
- Input is 21 bits
  - 1 constant bias bit =1
  - 5 bits that are set randomly on every example i.i.d.
  - 15 bits that change very slowly; one is selected at random, and flipped, every 10,000 steps
- The result is a slowly-changing target function in the 5 rapidly changing bits
- The learning network has the same structure, but learned weights and differential activation function, and only 5 hidden units

Systematic study of Backprop (SCR)

varying step size and activation functions

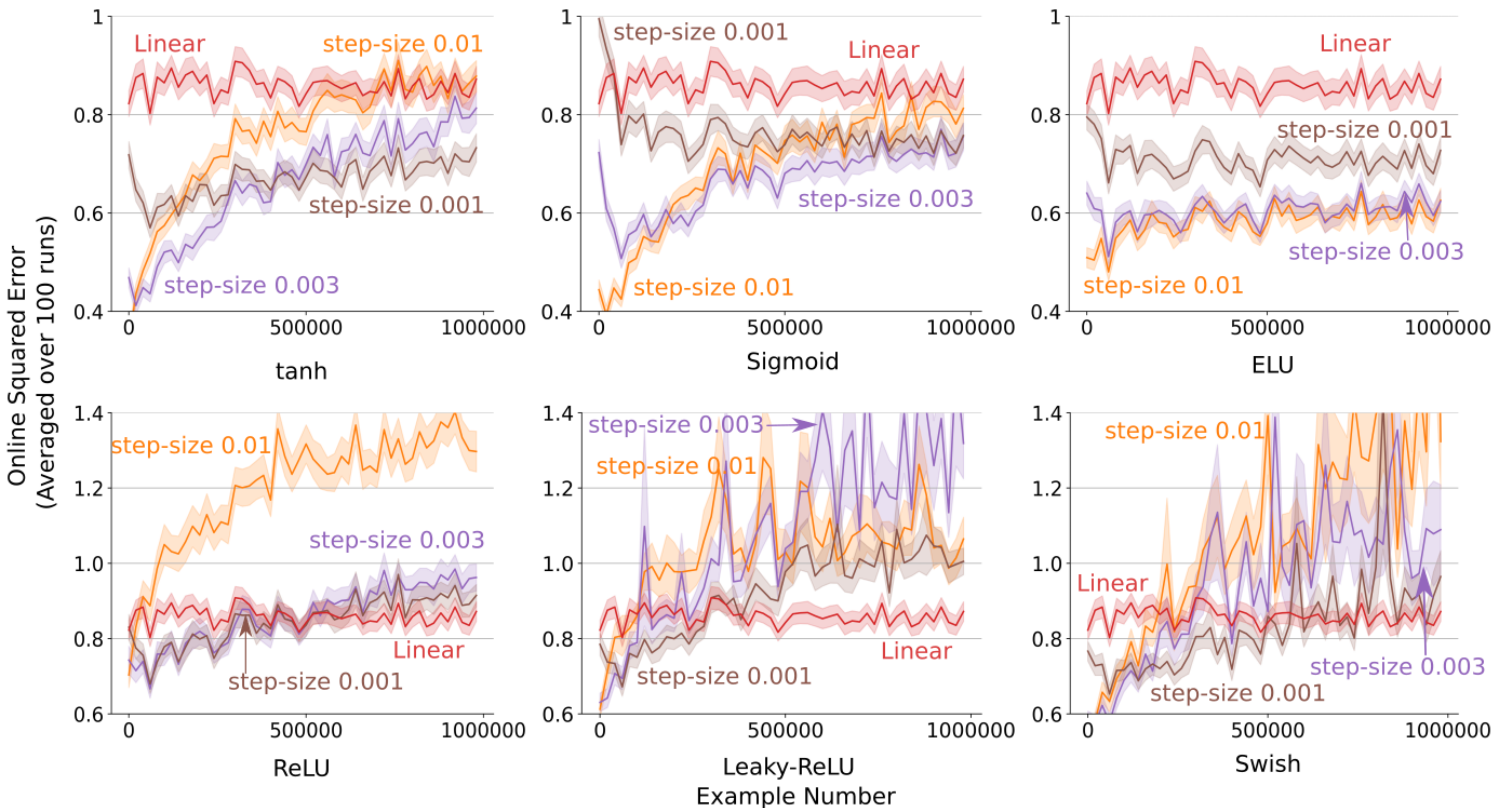


All algorithms vs All activations (SCR)



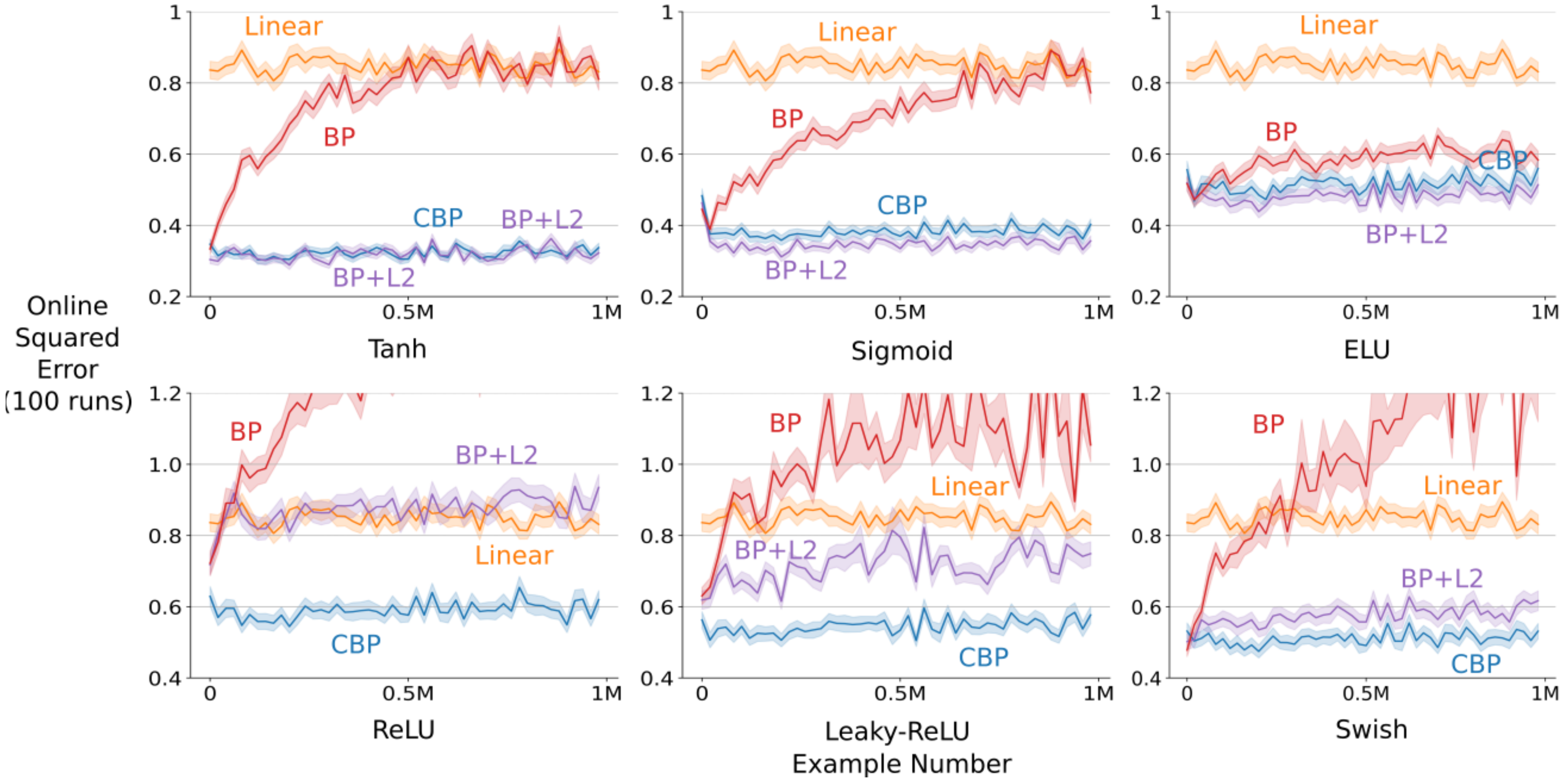
Systematic Study of Adam-Backprop (SCR)

varying both step size and activation function



Systematic Study of Adam-based algorithms (SCR)

and activation functions



Main message:

Deep learning does not work for continual learning

- by “not work” I mean that learning slows, eventually to a very low level
- by “deep learning” I mean the standard methods specialized to work in one-time learning

Better learning algorithms, specialized for continual learning, are not hard to find.

But we have to start looking for them, we have to get out of the rut!

Outline

- Demonstrations of Loss of Plasticity
  (in continual versions of ImageNet, MNIST, and generic regression)
- Understanding Loss of Plasticity
- Existing Methods that try to Maintain Plasticity
- A Simple Extension of Backprop, *Continual Backprop*,
  that Fully Succeeds in Maintaining Plasticity

Is Conventional Backpropagation continual?

- Conventional Backpropagation algorithm:
  - Initialization with small random weights
  - Gradient Descent at every time-step
- It is not a continual algorithm as it does a critical computation (initialization with small random weights) at the beginning which is not repeated again

How can we make the conventional backpropagation algorithm continual?

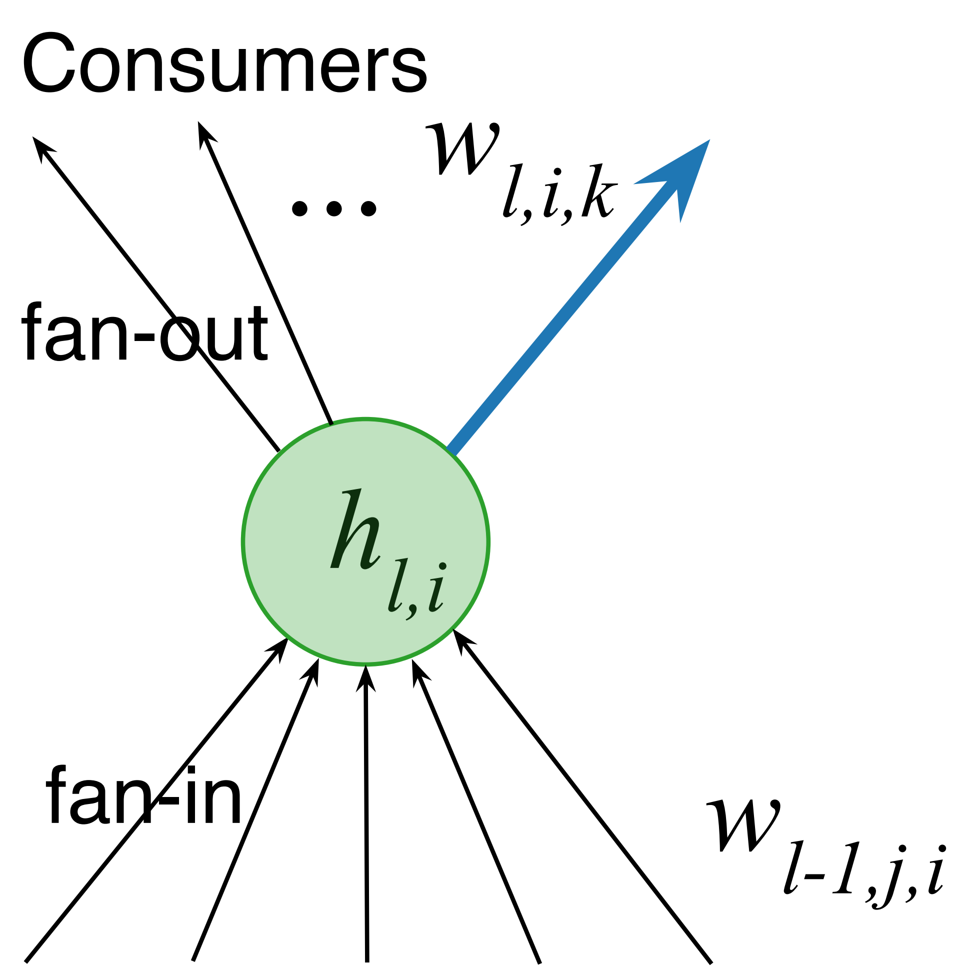
Continual Backprop:

Stochastic Gradient Descent with Selective Reinitialization

- We can just reinitialize occasionally, or a little bit at a time
- And it is better to reinitialize selectively, for example, dead units or some other notion of *utility*
- The idea of selective random initialization was introduced by Mahmood and Sutton (2012); they called it *generate and test*
- We extend the idea to general multi-layer networks

Continual Backprop:

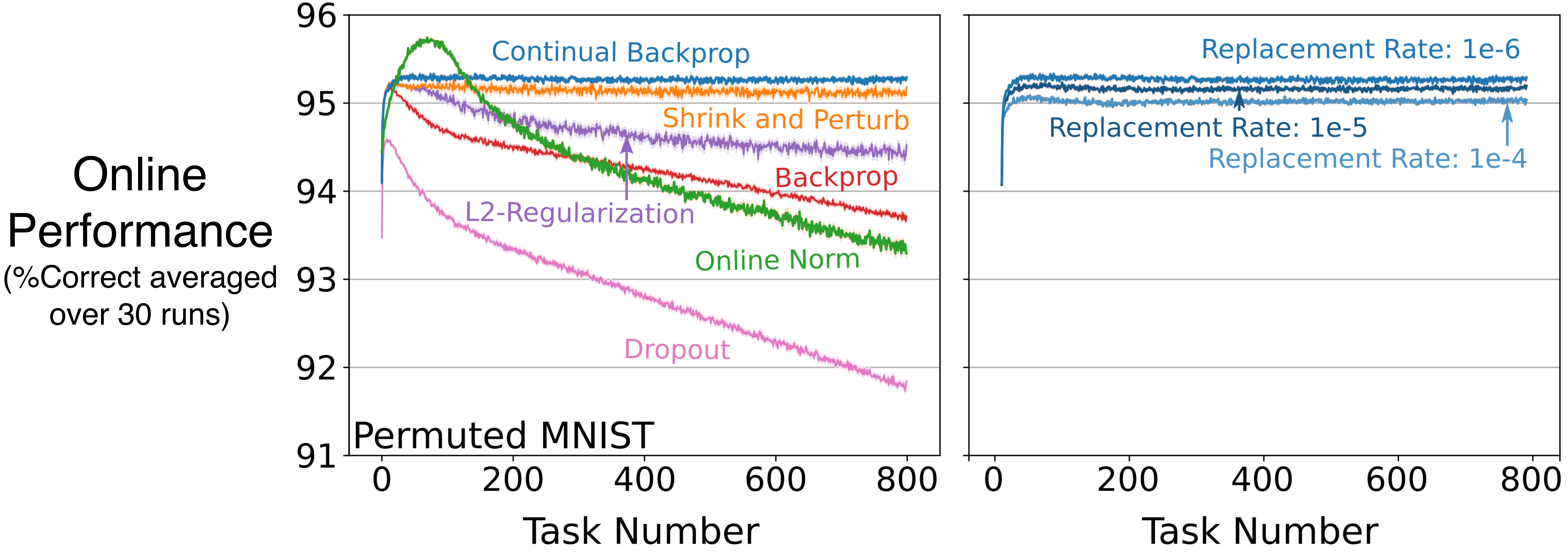
Stochastic Gradient Descent with Selective Reinitialization



$$y_{l,i,t} = \frac{|h_{l,i,t} - \hat{f}_{l,i,t}| * \sum_{k=1}^{n_{l+1}} |w_{l,i,k,t}|}{\sum_{j=1}^{n_{l-1}} |w_{l-1,j,i,t}|}$$

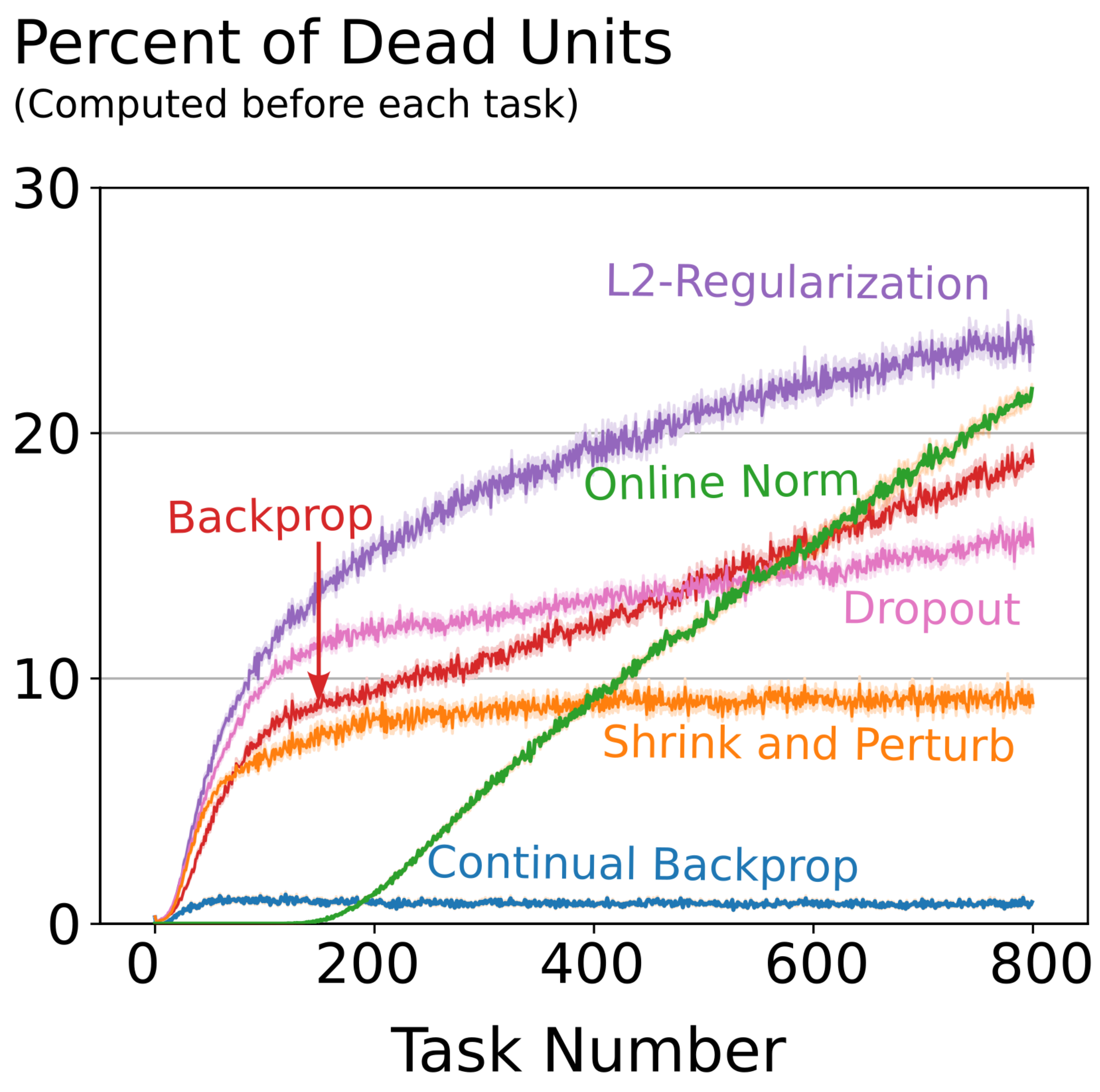
- Extensions to Mahmood's work:
  - General multi-layer networks where fan-out is greater than one
  - Consider feature activity in its utility instead of just the outgoing weights
  - Also consider how labile a feature is when computing its utility
  - Transfer a feature's average contribution to the bias of its consumers, so the consumers are less affected by the feature's removal
  - (Future direction) A global measure of utility instead of a local measure
  - (Future direction) Better generators (initializers)

Continual Backprop on online Permuted MNIST

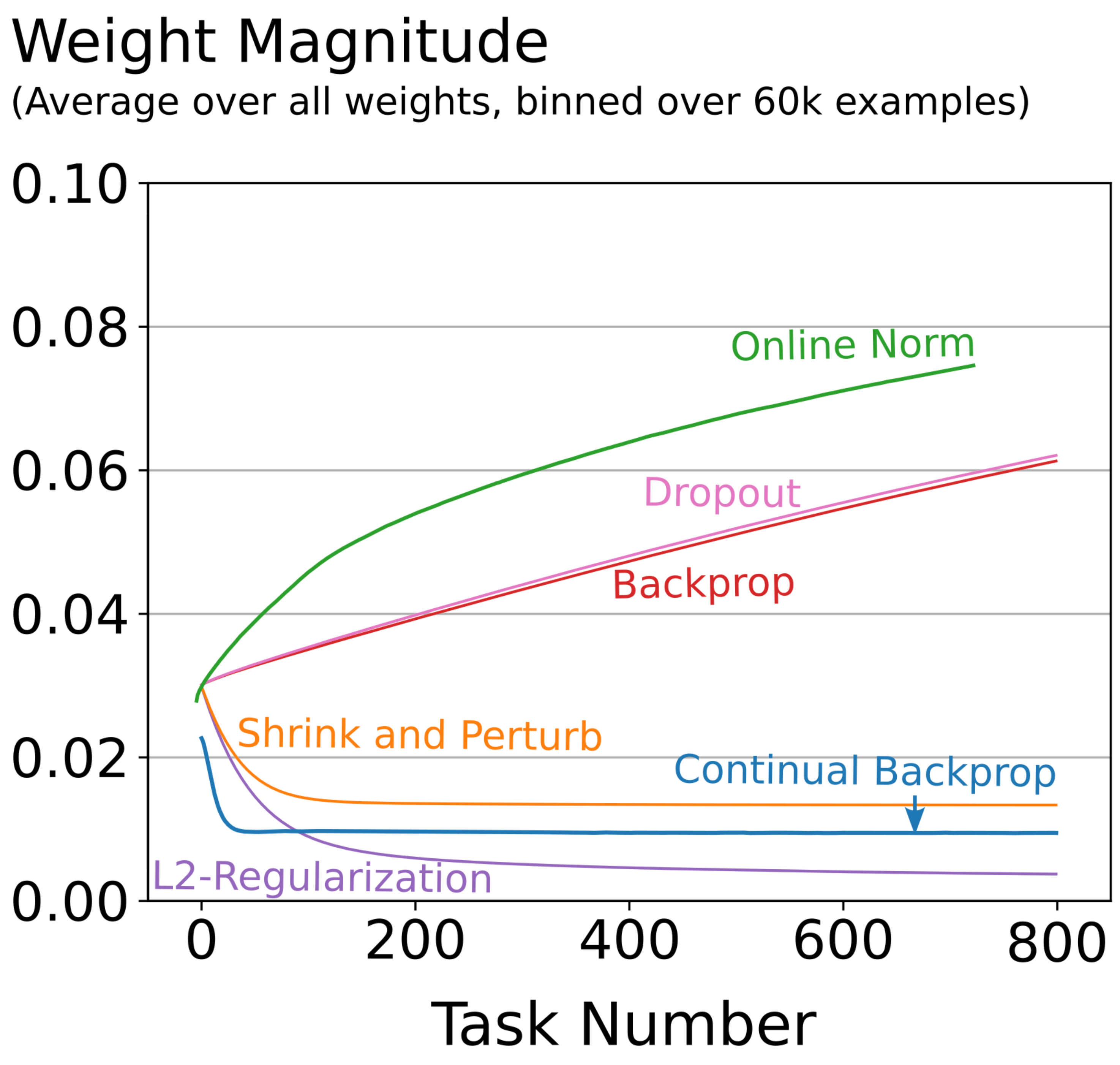


Continual Backprop fully maintains plasticity
and is fairly insensitive to its hyper-parameter, replacement-rate

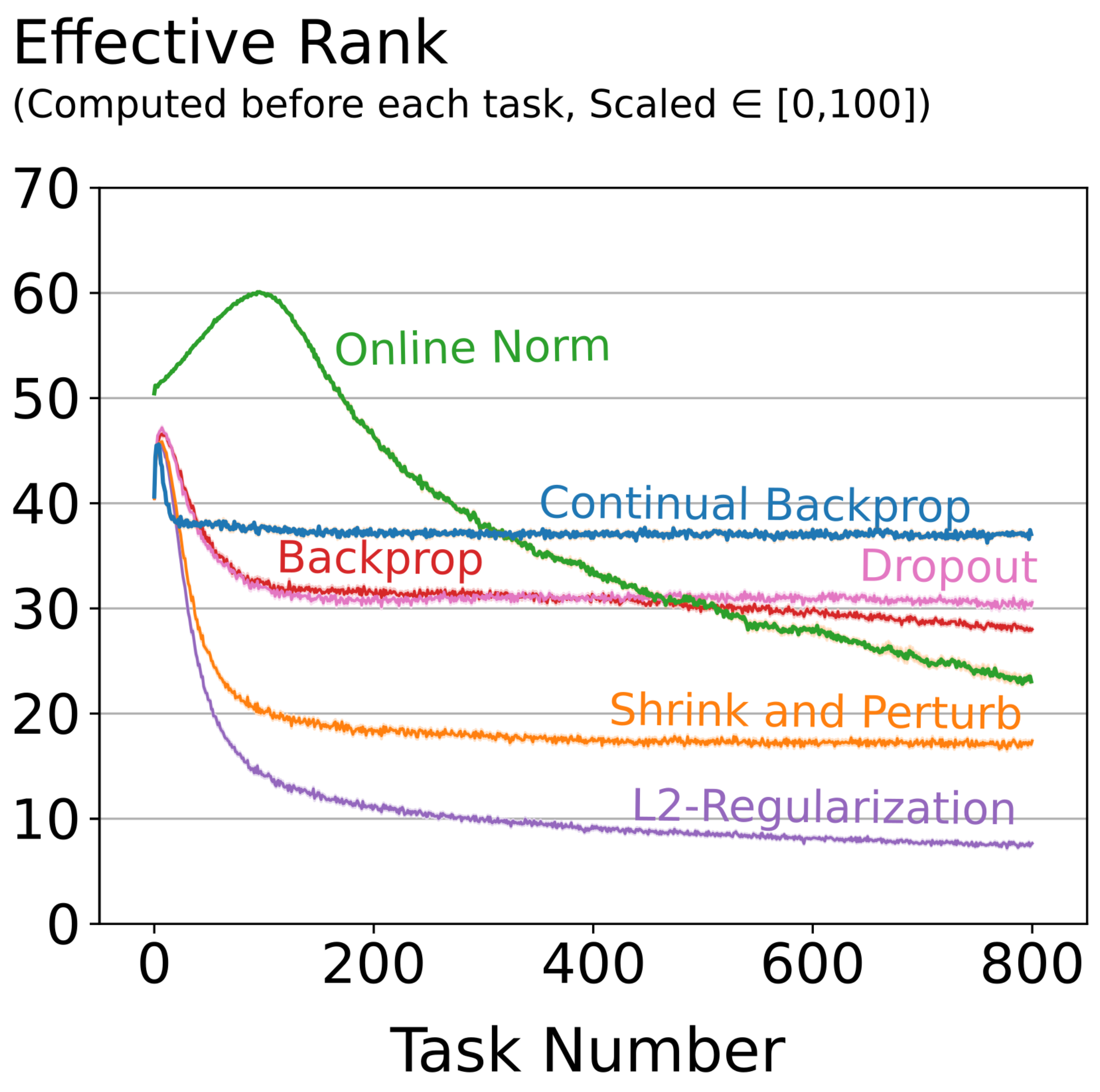
Understanding Continual Backprop (MNIST)



Continual Backprop has almost no dead units



Continual Backprop keeps the weights from getting too large



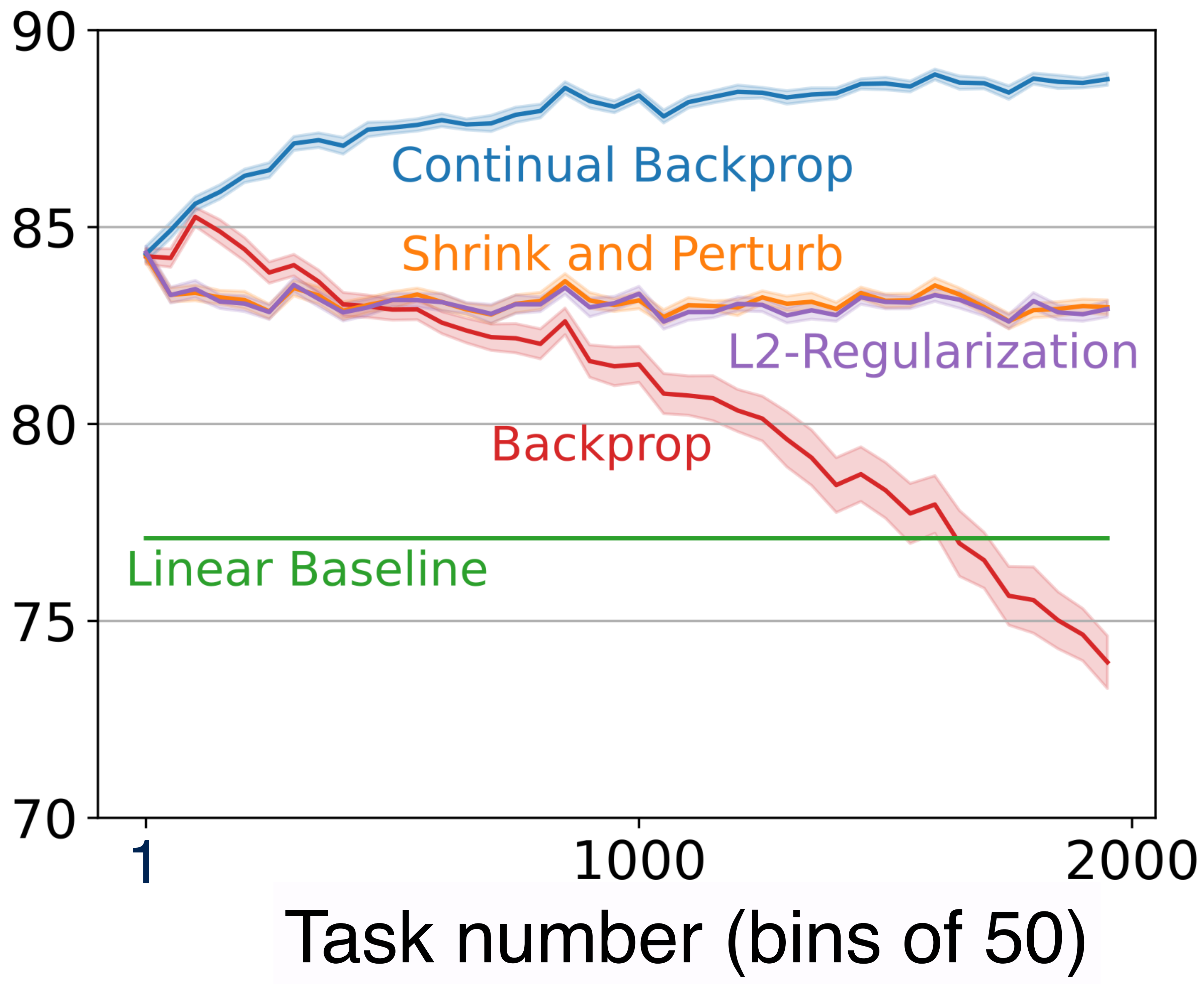
Continual Backprop maintains a high level of effective rank

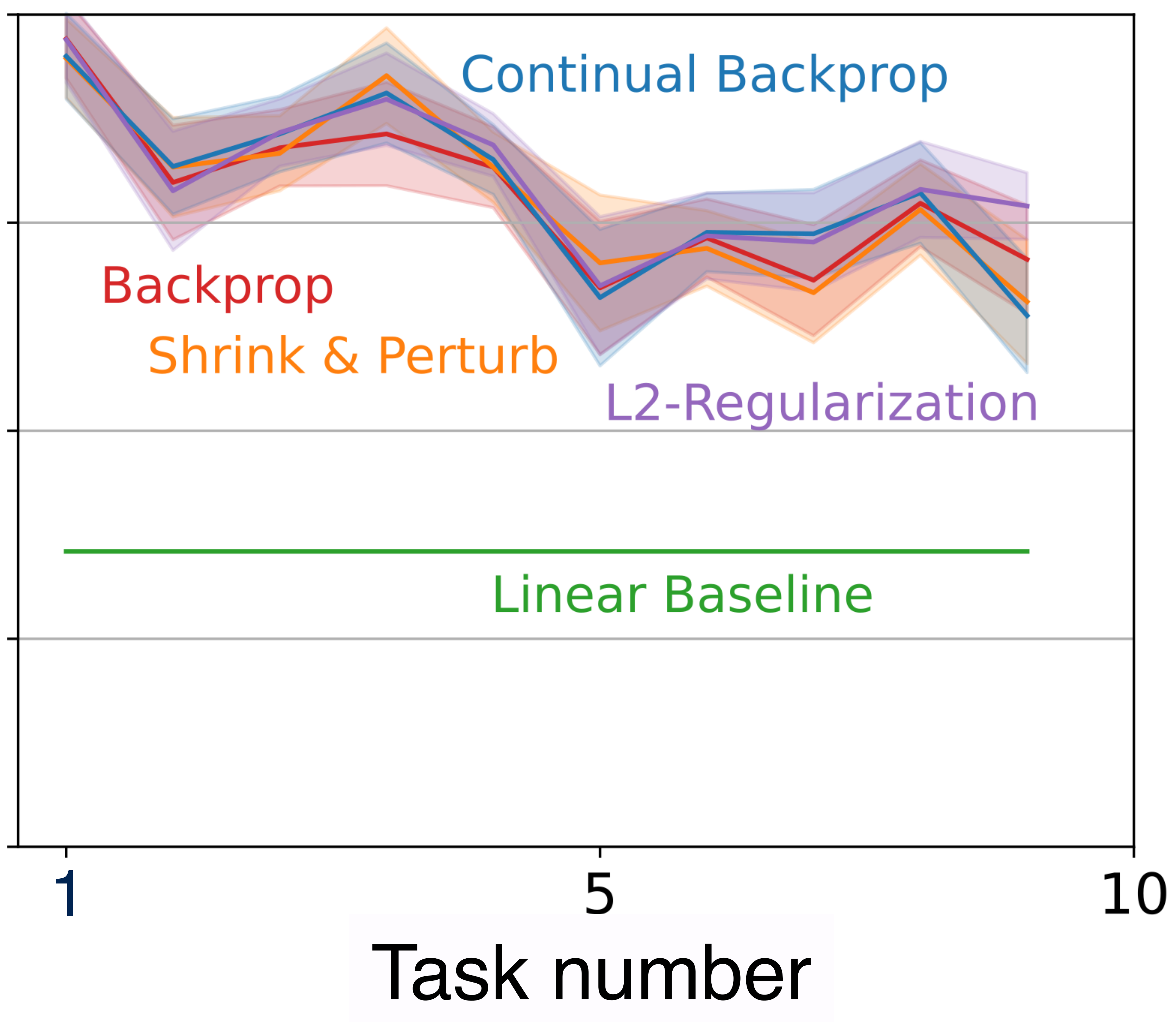
Continual Backprop seems to solve all the problems of LoP in MNIST

Continual Backprop on ImageNet

%Correct on Test Set

(at end of each task, averaged over 30 runs)



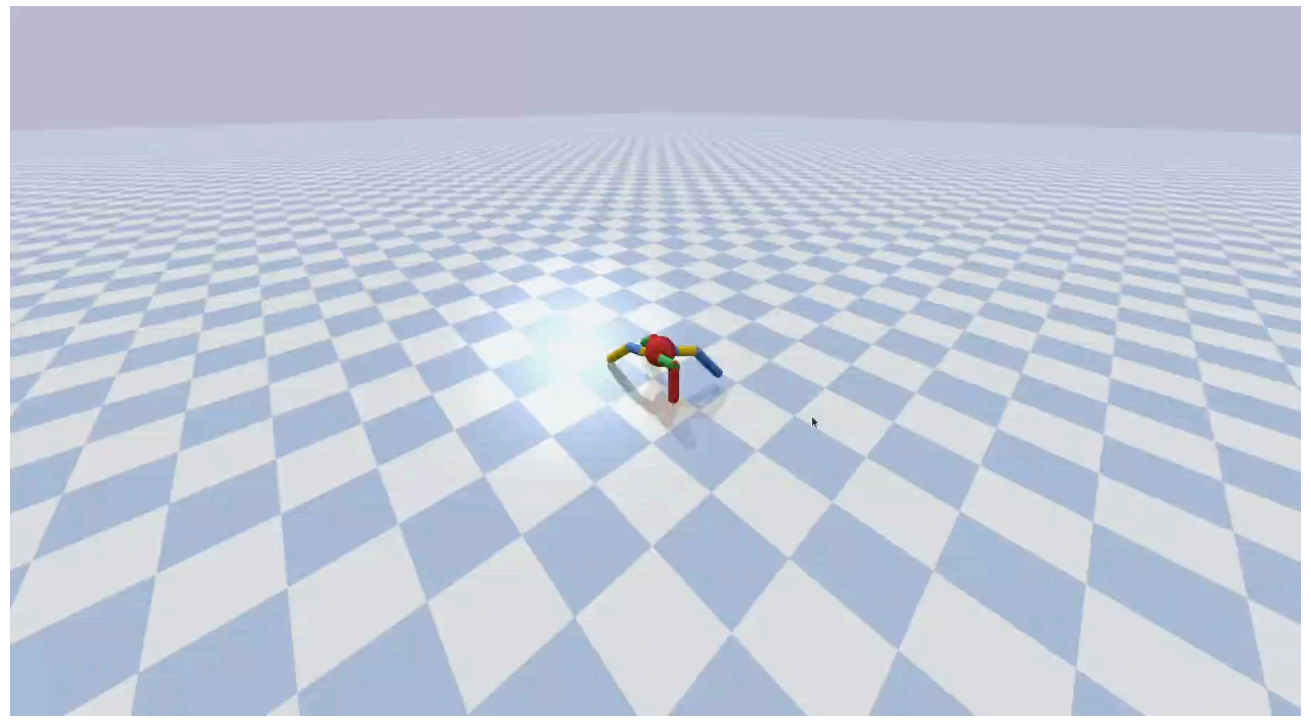


Continual Backprop fully maintains plasticity in ImageNet!

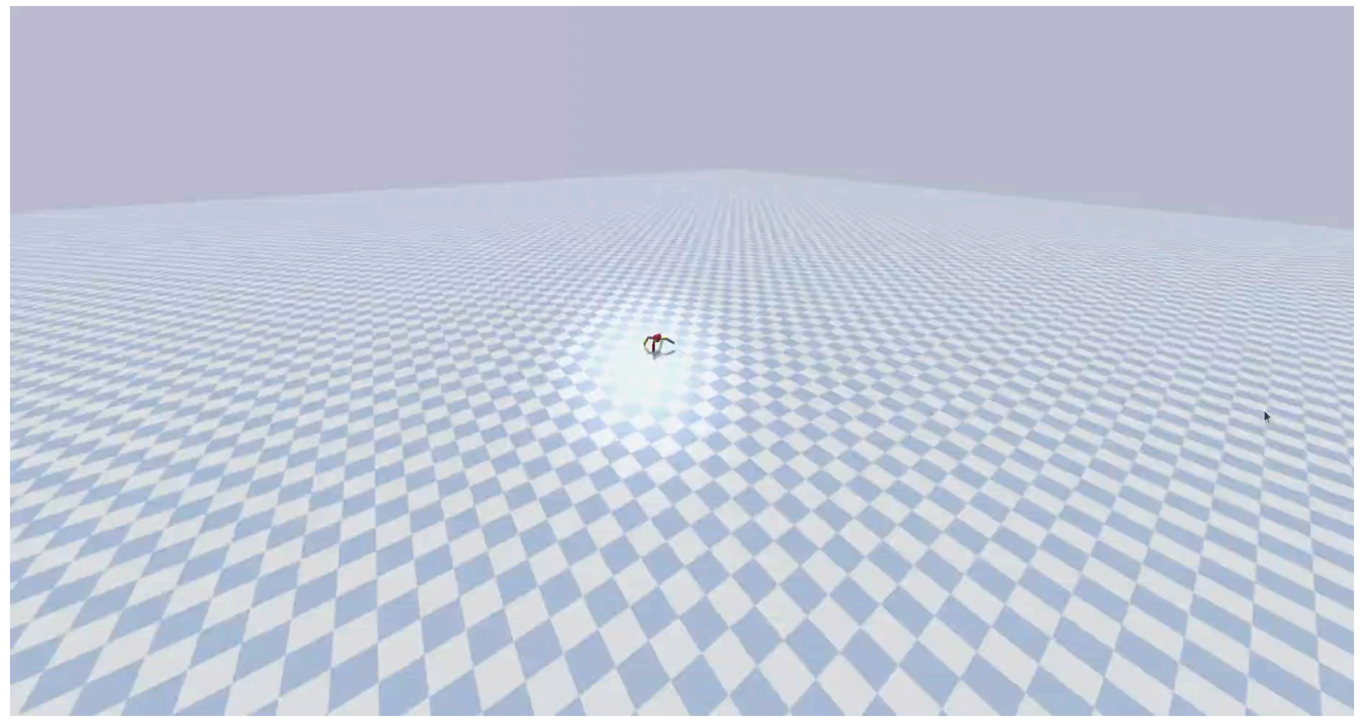
Extension to a non-stationary RL problem, *Slippery Ant*

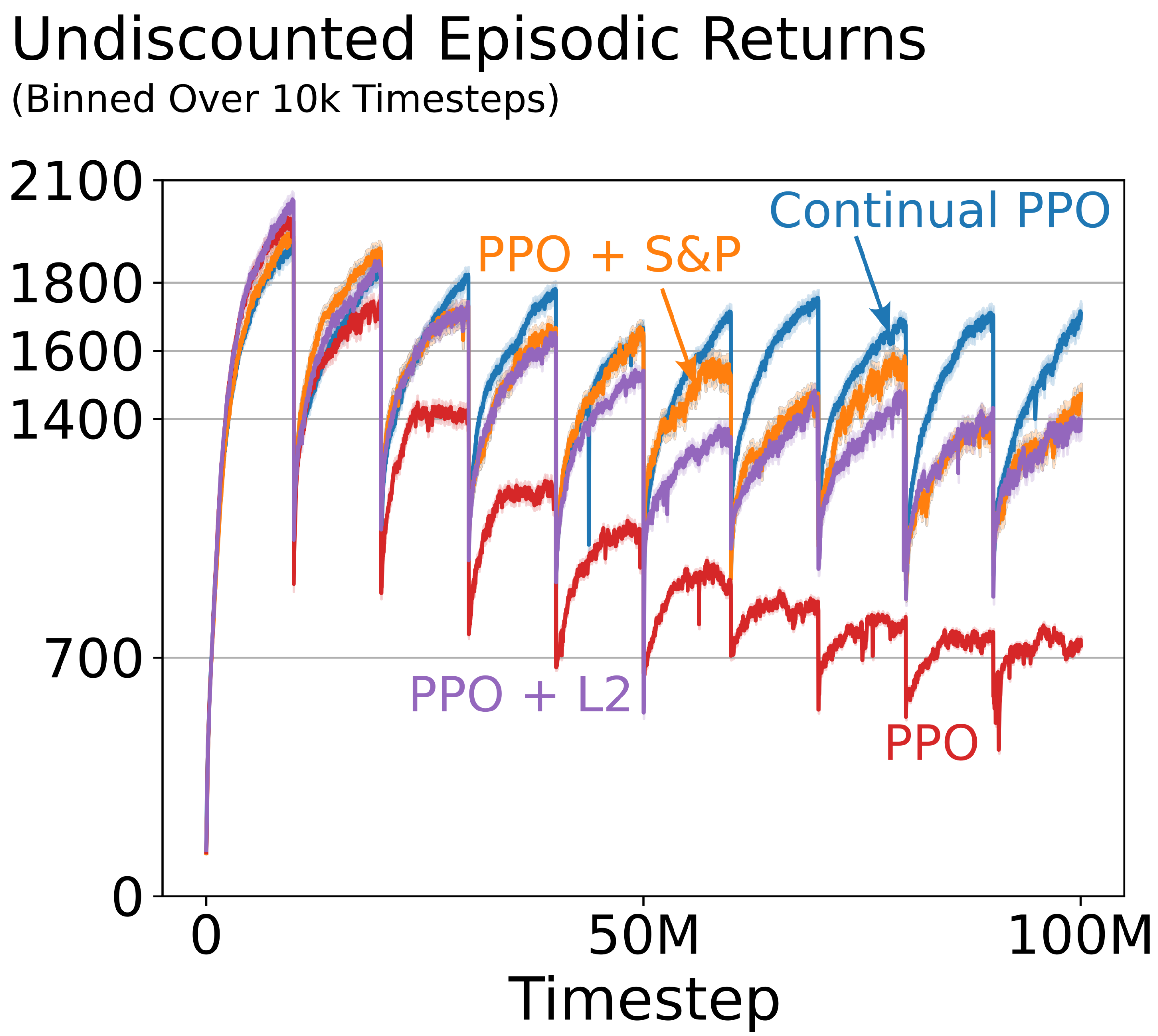
- An agent controlling a PyBullet ant is rewarded for forward movement
- The friction between with the ground changes every 10M time steps
- We use a *Continual* version of PPO

PPO



Continual PPO





PPO shows similar degradation in performance as Backdrop.

While Continual PPO maintains most (but not all) of its plasticity.

Conclusions

- Deep-learning networks are optimized for one-time learning,
and in a sense they totally fail for continual learning

- Simple changes, like Continual Backprop, can make them effective for continual learning

- Continual Backprop ranks units by their utility to the network's functioning. There are more improvements possible in how ranking is done, particularly for recurrent networks

- There is an exciting world ahead of deep-learning networks that can learn continually

- It opens up great new possibilities in RL (which is inherently continual due to policy iteration) and in advanced model-based RL architectures (which learn continually in multiple interacting components)

Thank you for your attention