Maintaining Plasticity in Deep Continual Learning

Marin Gennadiy

Definitions and Problems

Train-once vs Continual learning

Train-once - means that training occurs once on a large data set and then never again.

Continual learning - focuses on learning continually from new data

What if data distribution changes?

Train a new network from scratch

Training never stops (nothing changes)

Problems of Continual Learning

“Catastrophic forgetting” - deep learning systems, when exposed to new data, tend to forget most of what they have previously learned

Models lose their ability to keep learning from new data, will refers to this ability as **“losing plasticity”**

Maintaining plasticity is essential for continual learning systems as it allows them to adapt to changes in their data streams

Experiments on ImageNet

ImageNet for Continual Learning

Let’s create binary classification task for two different classes from ImageNet:

1) The 700 images from each class were divided into 600 images for training and 100 for testing.
2) Choose pair of classes randomly without replacements then train and test model on them (task)
3) After training and testing on one task, the next task begins based on a different pair of classes

Authors generated sequence of 2000 different tasks - as the result continual-learning problem **“Continual ImageNet”**

Results

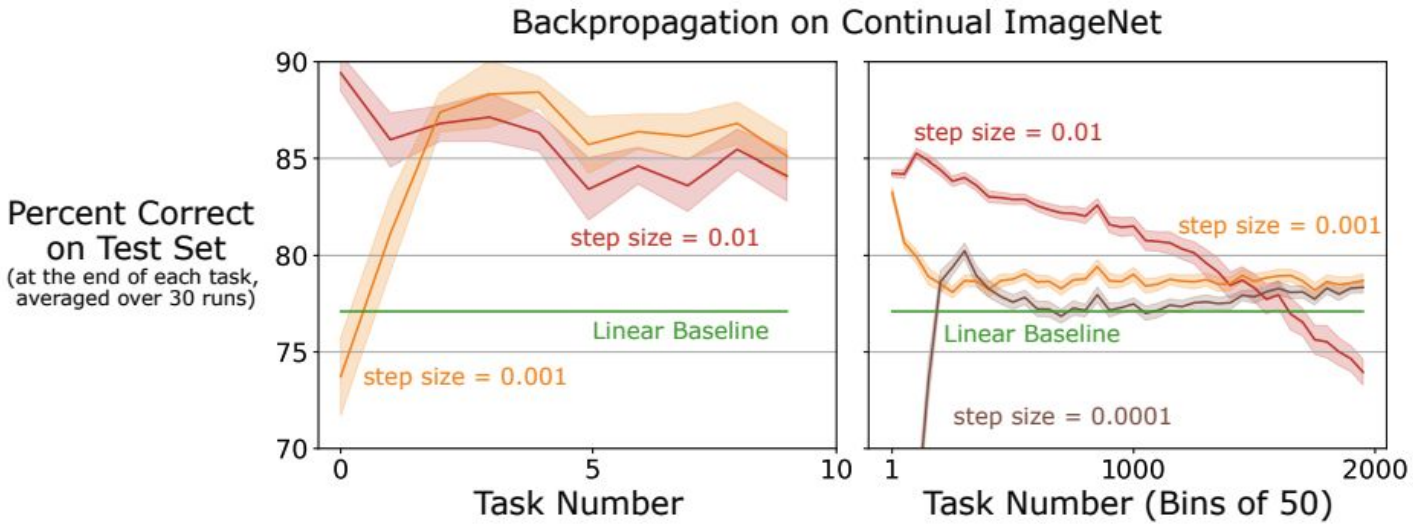


Figure 1: **Loss of plasticity on a sequence of ImageNet binary classification tasks**. The first plot shows performance over the first ten tasks, which sometimes improved initially before declining. The second plot shows performance over 2000 tasks, over which the loss of plasticity was extensive. The learning algorithm was backpropagation applied in the conventional deep-learning way.

Experiments on MNIST

MNIST for Continual Learning

1) Choose permutation and apply it to all 60’000 images (permuted MNIST)
2) Create a sequence of 800 permuted MNIST
3) For each task, present each of its images one-by-one in random order to learning network, then move to the next permuted MNIST task

Authors used a sequence of 800 tasks, gave no indication at the time of task switching, all images in one task were passed through network simultaneously - **“Online Permuted MNIST”**

Results

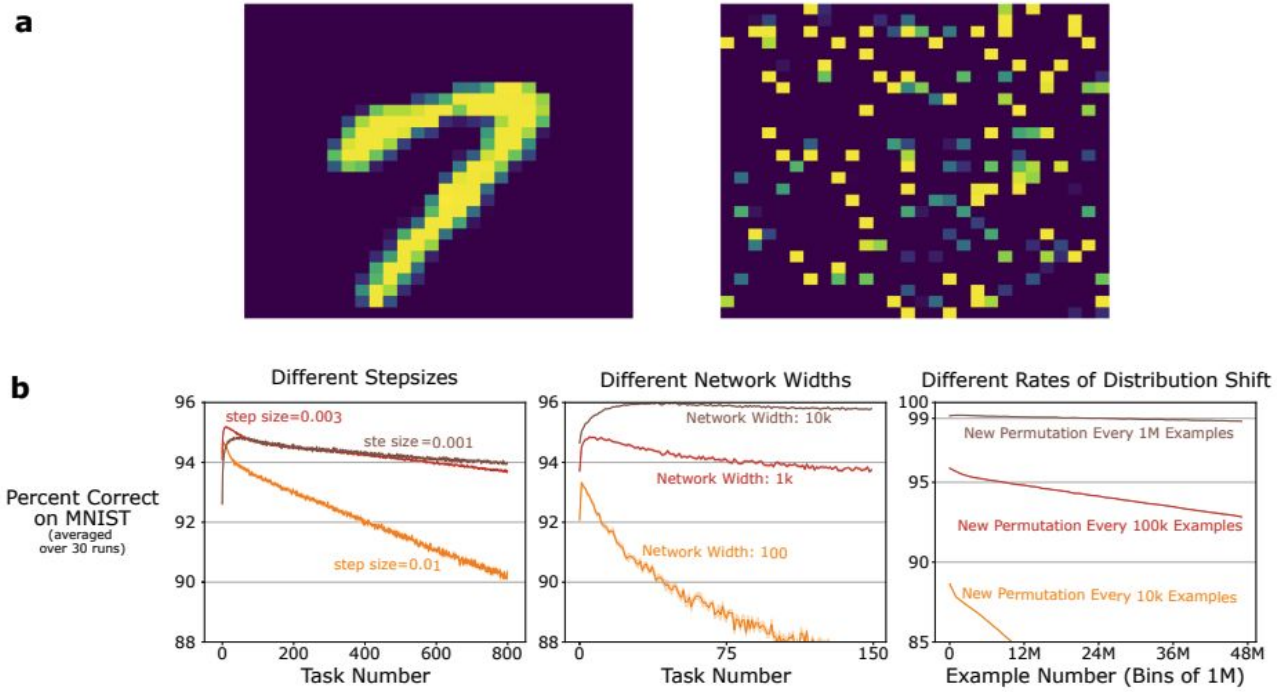


Figure 2: **a:** Left: An MNIST image with the label '7'; Right: A corresponding permuted image. **b:** Loss of plasticity in Online Permuted MNIST is robust over step sizes, network sizes, and rates of change.

Experiments conclusion

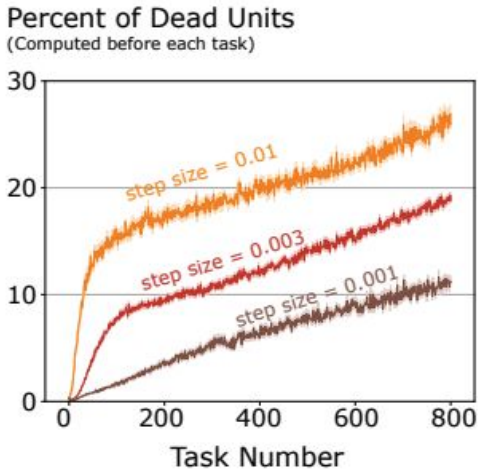
In continual learning conventional backpropagation doesn't work

Were performed direct tests of loss plasticity with backpropagation.

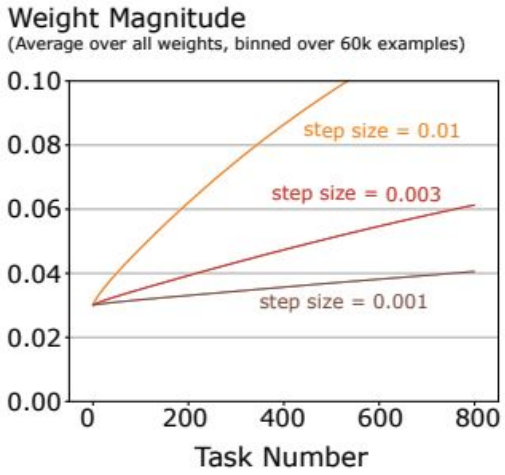
Experiments show that loss of plasticity is a general phenomenon.

Why backpropagation doesn’t work?

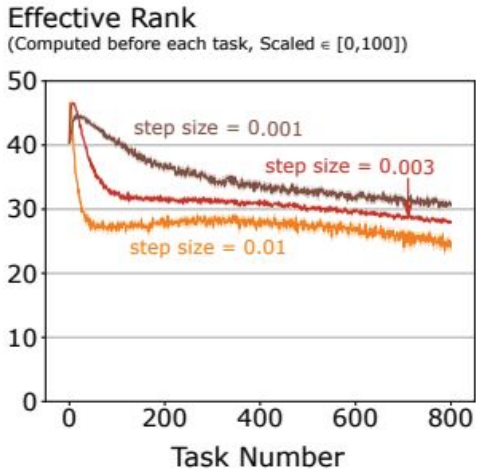
Possible reasons of plasticity loss during BP



severely slowing down learning



often associated with learning instability



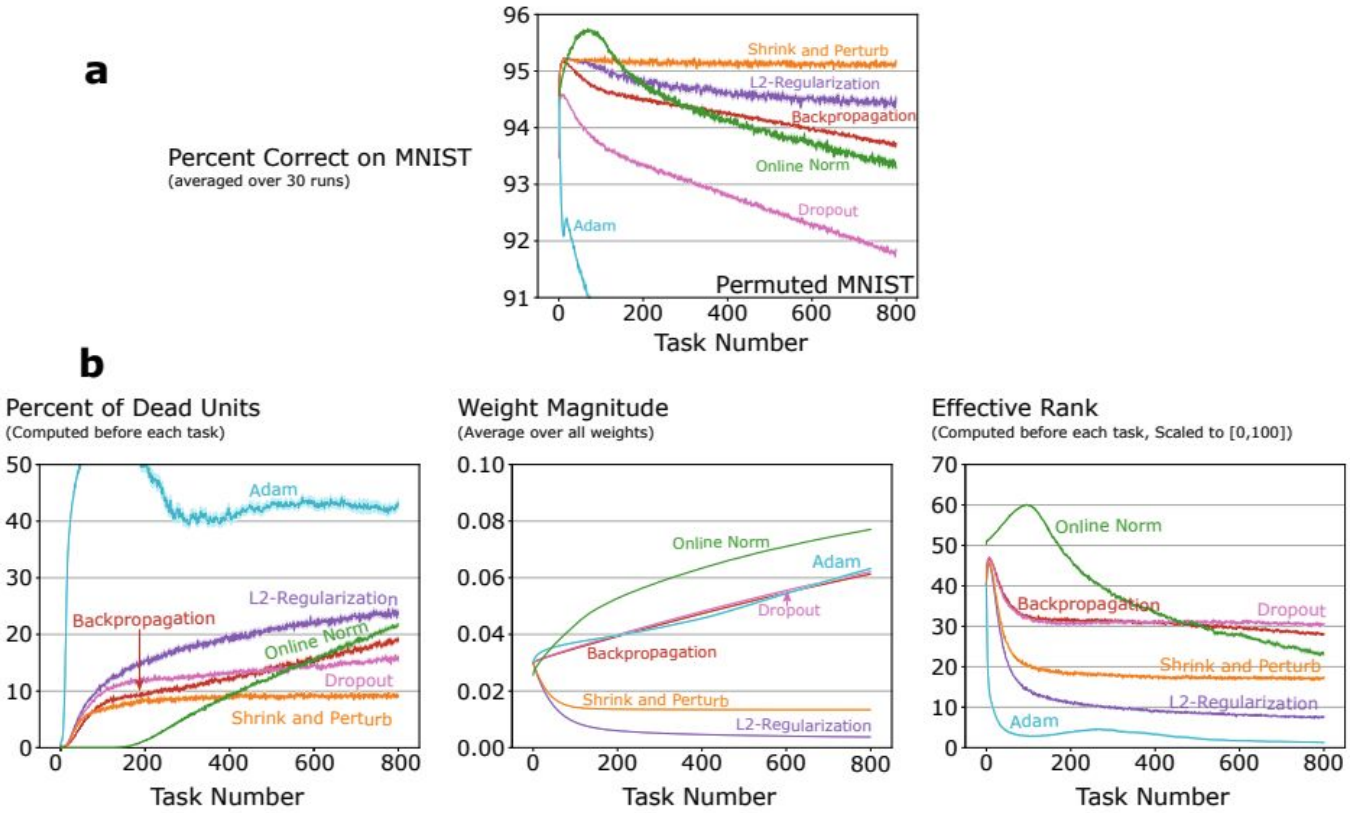
hidden units provide little to no information

Test current methods for mitigating loss of plasticity

Methods

- *L2-regularization* - directly reduces weight magnitude
- *Shrink-and-perturb* - shrink reduces weight magnitude and perturb (adds random noise) might help decrease number of dead units and increase effective rank
- *Dropout* - might increase the effective rank
- *Batch normalization* - is expected to mitigate problem of dead units
- *Adam* - works well with non-stationary losses, might work with non-stationary continual learning problems

Results


a
Percent Correct on MNIST
(averaged over 30 runs)
Shrink and Perturb
L2-Regularization
Backpropagation
Online Norm
Dropout
Adam
Permuted MNIST
Task Number
b
Percent of Dead Units
(Computed before each task)
Weight Magnitude
(Average over all weights)
Effective Rank
(Computed before each task, Scaled to [0,100])
Task Number

Modern methods conclusion

None of it solves three correlations and totally reduces plasticity loss

Adam, dropout worse the loss of plasticity

BatchNorm deteriorate measures over time

L2 and SnP reduce loss of plasticity (not totally remove), in addition, both are very sensitive to hyperparameter values

Continual Backpropagation:
SGD with Selective Reinitialization

Ideas

Continual backpropagation makes convention backpropagation continual by performing similar computations at all times

CBP selectively initializes low-utility units in the network:

1. Find low-utility units
    1.1. Contribution utility
    1.2. Adaptation utility
2. Reinitialize them

Reinitialization

- Every time step, a fraction of hidden units “ρ”, called *replacement-rate*, are reinitialized in every layer
- When a new hidden unit is added, its outgoing weights are initialized to zero, input weights are random
- To protect new units from immediate reinitialization (because outgoing weights are zeros), protect them for *maturity threshold* “m” numbers of updates

Utility: Contribution

Intuition: magnitude of the product of units' activation and outgoing weights gives information about how valuable this connection to its consumers

$$c_{l,i,t} = \eta * c_{l,i,t-1} + (1 - \eta) * |h_{l,i,t}| * \sum_{k=1}^{n_{l+1}} |w_{l,i,k,t}|,$$

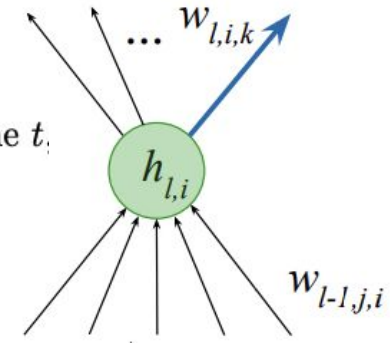
$c_{l,i,t}$, of the ith hidden unit in layer l at time t

$h_{l,i,t}$ is the output of the i^{th} hidden unit in layer l at time t

$w_{l,i,k,t}$ is the weight connecting the i^{tl} unit in layer l to the k^{th} unit in layer $l+1$ at time t

n_{l+1} is the number of units is layer $l+1$

decay rate, η.



Utility: mean-corrected Contribution

Thus hidden layers must be replaced while learning, it is needed to consider the learning process's effect on a unit's contribution, so we need to remove part of contribution which is correlated with the bias

mean-corrected contribution utility, $z_{l,i,t}$

$$f_{l,i,t} = \eta * f_{l,i,t-1} + (1 - \eta) * h_{l,i,t},$$

(4) $$\hat{f}_{l,i,t} = \frac{f_{l,i,t-1}}{1 - \eta^{a_{l,i,t}}},$$

(5) $$z_{l,i,t} = \eta * z_{l,i,t-1} + (1 - \eta) * |h_{l,i,t} - \hat{f}_{l,i,t}| * \sum_{k=1}^{n_{l+1}} |w_{l,i,k,t}|,$$

$f_{l,i,t}$ is a running average of $h_{l,i,t}$ and $\hat{f}_{l,i,t}$ is the bias-corrected estimate.

Utility

Adaptation utility is the inverse of the average magnitude of the units' input weights

Intuition: tries to capture how fast a hidden unit can get the function it is representing

Overall utility û:

$$y_{l,i,t} = \frac{|h_{l,i,t} - \hat{f}_{l,i,t}| * \sum_{k=1}^{n_{l+1}} |w_{l,i,k,t}|}{\sum_{j=1}^{n_{l-1}} |w_{l-1,j,i,t}|} \quad (6)$$

$$u_{l,i,t} = \eta * u_{l,i,t-1} + (1 - \eta) * y_{l,i,t},$$

$$\hat{u}_{l,i,t} = \frac{u_{l,i,t-1}}{1 - \eta^{a_{l,i,t}}}.$$

Algorithm 1: Continual backpropagation (CBP) for a feed-forward network with L hidden layers

Set: step size α, replacement rate ρ, decay rate η, and maturity threshold m (e.g. 10^{-4}, 10^{-4}, 0.99, and 100)

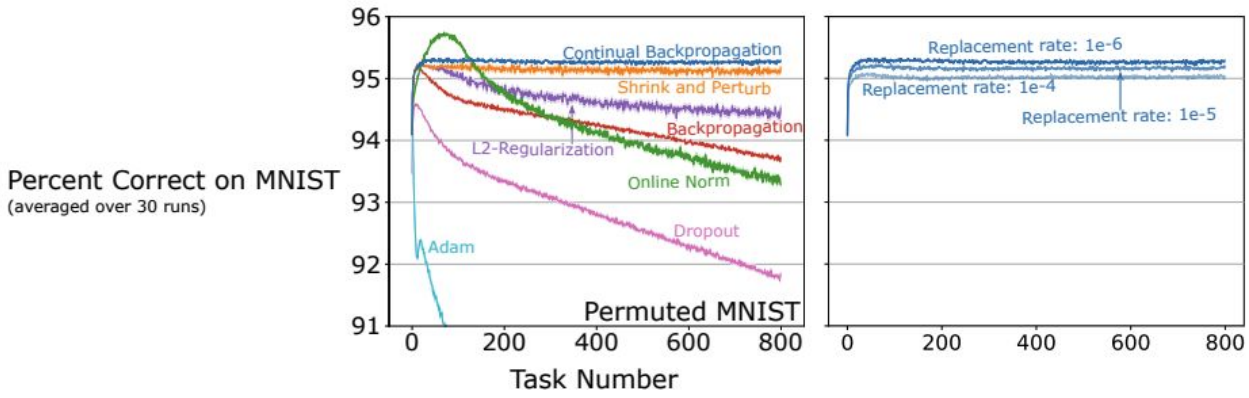
Initialize: Initialize the weights $\mathbf{w}_0, ..., \mathbf{w}_L$. Let, $\mathbf{w}_l$ be sampled from a distribution d_l

Initialize: Utilities $\mathbf{u}_1, ..., \mathbf{u}_L$, average activation $\mathbf{f}_1, ..., \mathbf{f}_l$, and ages $\mathbf{a}_1, ..., \mathbf{a}_L$ to 0

for *each input* x_t **do**

- **Forward pass:** pass input through the network, get the prediction, $\hat{y}_t$
- **Evaluate:** Receive loss $l(x_t, \hat{y}_t)$
- **Backward pass:** update the weights using stochastic gradient descent
- **for** *layer* l *in* $1 : L$ **do**
  - **Update age:** $\mathbf{a}_l \mathrel{+}= 1$
  - **Update unit utility:** Using Equations 4, 5, and 6
  - **Find eligible units:** Units with age more than m
  - **Units to reinitialize:** $n_l * \rho$ of eligible units with the smallest utility, let their indices be $\mathbf{r}$
  - **Initialize input weights:** Reset the input weights $\mathbf{w}_{l-1}[\mathbf{r}]$ using samples from d_l
  - **Initialize output weights:** Set $\mathbf{w}_l[\mathbf{r}]$ to zero
  - **Initialize utility, unit activation, and age:** Set $\mathbf{u}_{l,\mathbf{r},t}$, $\mathbf{f}_{l,\mathbf{r},t}$, and $\mathbf{a}_{l,\mathbf{r},t}$ to 0

Results


a
Percent Correct on MNIST
(averaged over 30 runs)
Continual Backpropagation
Shrink and Perturb
Backpropagation
L2-Regularization
Online Norm
Dropout
Adam
Permuted MNIST
Task Number
Replacement rate: 1e-6
Replacement rate: 1e-4
Replacement rate: 1e-5
b
Percent of Dead Units
(Computed before each task)
Weight Magnitude
(Average over all weights)
Effective Rank
(Computed before each task, Scaled to [0,100])

Results

Fully mitigates loss of plasticity in Continual ImageNet

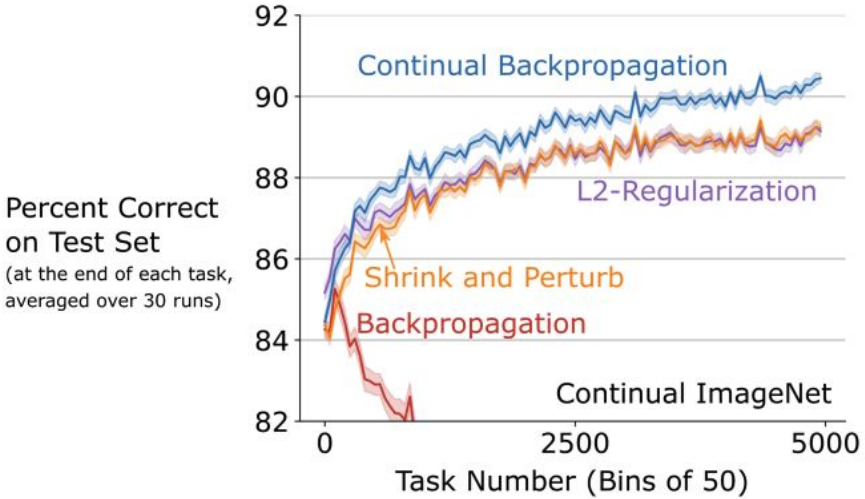


Figure 7: Continual backpropagation outperforms all the existing algorithms and fully maintains plasticity on Continual ImageNet. Its performance at the end of 5000 tasks is even better than on the first task.

Continuous Backpropagation

Mitigates all three correlates of loss of plasticity

Only CBP has non-degrading performance

CBP staple for a wide range of hyperparameter values

Additionally, in contrast to shrink-and-perturb CBP injects randomness selectively , effectively removing all dead units from the network, leading to higher effective rank

Conclusion

We established in a more definitive way that deep learning methods lose the ability to learn in continual-learning

Discovered factors which correlate with plasticity loss

Introduced a new algorithm (continuous backpropagation) which solves plasticity loss problem

Thanks for your attention!