# Continual Learning with Group-wise Neuron Competition Appendix

**Anonymous Author(s)**
Affiliation
Address
email

In the following, we provide the Appendix as part of the supplementary material to the main paper.

**In Appendix Section A.**, we prove the alternative form for the back-propagation to make more obvious at which part the gradient of the activation with respect to it arguments comes into use. **In Appendix Section B.**, we give additional comment about applying group-wise neuron normalization with nonlinear activation, providing additional results. **In Appendix Section C.**, we prove additional result regarding performance comparison of existing normalization techniques and our GNC. **In Appendix Section D.**, we give very simple pythorch code implementation of the proposed group-wise neuron normalization that induces group-wise neuron competition (GNC).



Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

# A. Back-propagation Alternative Form

In the main text of the paper, we analyzed the influence of GNC that has during back-propagation, so we were interested in the gradient of the activation with respect to its arguments, *i.e.*, $\mathbf{g}^{L-1} = \frac{\partial \mathbf{a}^{L-1}}{\partial \mathbf{z}^{L-1}}$. In this appendix section, we give the full derivation of the equivalent form for the back-propagation, and pinpoint where $\frac{\partial \mathbf{a}^{L-1}}{\partial \mathbf{z}^{L-1}}$ comes to play.

Consider a neural network that maps inputs $\mathbf{x} \in \mathbb{R}^{N_0}$ to outputs $\mathbf{y} \in \mathbb{R}^{N_L}$. The forward pass reads as:

$$\begin{aligned} \mathbf{a}^0 &= \mathbf{x}, \\ \mathbf{z}^l &= \mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l, \qquad l \in \{1, \cdots, L\}, \\ \mathbf{a}^l &= \sigma(\mathbf{z}^l), \qquad l \in \{1, \cdots, L\}. \end{aligned} \tag{1}$$

Here, $\mathbf{W}^l \in \mathbb{R}^{N_l \times N_{l-1}}$ is the weight matrix of layer $l$ (note that if we have convolution, we can express it similarly as a matrix product multiplication), $\mathbf{b}^l$ the corresponding bias vector. Where $N_l$ denotes the dimension of the layer $l$. The activation function is denoted by $\sigma$, it processes the layer's weighted sum $\mathbf{n}^l$ to the layer's output $\mathbf{a}^l$.

Given data set, we define the training process of a neural network as an optimization over an objective function, usually called *loss*, that we denote as $\mathcal{L}$. In other word, we want to find the parameters (weights and biases) that minimize the cost function. The training is commonly done using a gradient based rule. Therefore, the update relies on the gradient of $\mathcal{L}$ with respect to weight $\mathbf{W}^l$ and the bias $\mathbf{b}^l$, that is it relies on $\nabla_{\mathbf{W}}\mathcal{L}$ and $\nabla_{\mathbf{b}}\mathcal{L}$, respectively. Back-propagation facilitates the computation of these gradients, and makes use of the chain rule to back-propagate the prediction error through the network [7]. We express the error vector at layer $l$ as:

$$\delta^l = \nabla_{\mathbf{n}^l}\mathcal{L}, \tag{2}$$

and further use it to express the gradients as

$$\begin{aligned} \nabla_{\mathbf{W}^l}\mathcal{L} &= \delta^l (\mathbf{a}^{l-1})^{\mathrm{T}}, \\ \nabla_{\mathbf{b}^l}\mathcal{L} &= \delta^l. \end{aligned} \tag{3}$$

The output layer's error is simply given by

$$\delta^L = \nabla_{\mathbf{a}^L}\mathcal{L} \odot \frac{\partial \mathbf{a}^L}{\partial \mathbf{z}^L}, \tag{4}$$

where $\odot$ denotes the Hadamard element-wise product. Subsequent earlier layer's error are computed with

$$\delta^l = (\mathbf{W}^{l+1})^{\mathrm{T}} \delta^{l+1} \odot \frac{\partial \mathbf{a}^l}{\partial \mathbf{z}^l}, \qquad l \in \{1, \cdots, L-1\}. \tag{5}$$

where A usual parameter update takes on the form

$$(\mathbf{W}^l)_{\text{new}} = \mathbf{W}^l - \beta \nabla_{\mathbf{W}^l}\mathcal{L}, \tag{6}$$

where $\beta$ is a positive learning rate.

# B. Additional Comment on Group-wise Neuron Competition

## B.1. Relevance of the Order for the Operations Sparse Coding and GNC

Table 1: Results using the CIFAR100 data set in the single pass continual learning regime. At the penultimate layer we used two different operating orders (i) GNC followed by sT and (ii) sT followed by GNC, showing that the difference between the two is negligible.

| sT Activation Followed by GNC | | GNC Followed by sT Activation | |
|---|---|---|---|
| RA | BTI | RA | BTI |
| **61.47**$\pm$1.07 | -04.34$\pm$-1.34 | 61.30$\pm$0.56 | -**04.07**$\pm$-0.94 |

**Existing Methods (Normalization Followed by Nonlinear Activation).** In the existing normalization techniques like batch norm [4], group norm [9] and layer norm [2], it is common that we first apply the normalization and then the nonlinear activation, *i.e.*, $\mathbf{a}^l = ReLu(bn(\mathbf{z}^l))$, where $bn(\mathbf{z}^l)$ is the batch norm in this example. It turn out that when we use the existing normalization techniques [4, 9, 2] it makes a big difference whether (i) we fist apply the nonlinear activation and then the apply the normalization, or (ii) we fist apply the normalization and then apply the nonlinear activation. In the later case, using normalization with learnable parameters [4, 9, 2], we might have degradation of performance. We mention this in the paper as well as give additional results in Section C.

**GNC.** In our approach we found out that it dose not play a role if we fist apply the nonlinear activation (at least in the case of sT) and then the GNC, or the other way around. In the following we will revisit the definition about GNC and give explanation.

In the main text, we construct $\mathbf{a}^l$ as $\mathbf{a}^l = [(\mathbf{a}^l_1)^T, (\mathbf{a}^l_2)^T, ..., (\mathbf{a}^l_J)^T]^T$ and define the group-wise normalization as:

$$\mathbf{a}^l_j = v_{GNC}(\mathbf{z}^l_j) = \frac{\mathbf{z}^l_j}{\|\mathbf{z}^l_j\|}, \forall j \in \{1, .., J\}, \tag{7}$$

where $\mathbf{z}^l_j$ are linear activation's that belong to group $j$ and $\|\mathbf{z}^l_j\|$ is the $\ell_2$-norm of $\mathbf{z}^l_j$. We also mentioned that before the normalization, we can apply nonlinear activation, such as ReLu [1], so we have $\mathbf{a}^l_j = v_{GNC}(\text{ReLu}(\mathbf{z}^l_j))$, or with sparse coding transform sT [6] we have:

$$\mathbf{a}^l_j = v_{GNC}(\text{sT}(\mathbf{z}^l_j)). \tag{8}$$

Note that the group-wise neuron competition is about the max magnitude neuron response that should lay in the group with the highest sparsity. Therefore, when we apply the sT, we pronounce the sparsity (ether before or after) within the group of neuron responses, and this should help the group-wise neuron competition, since we remove (threshold out) some not-relevant "information".

In Tab. 1, we give results that evaluated the above line of tough. We experimented using the CIFAR100 data set in the task-aware single pass regime. We set the reservoir size of 200 and use a reservoir batch size of 20. As we can see in Tab. 1, the results suggest that the order of these operation dose not play a role. When we use the sT after the GNC, very slightly we have higher retained accuracy, while when we use the sT before the GNC, very slightly we have better backward transfer. So, the difference in results between the two are negligible.

### B.2. Soft Sparse Coding Transform vs Hard Sparse Coding Transform

Table 2: Results using the CIFAR100 data set in the single pass continual learning regime. At the penultimate layer we used two different sparse coding transforms (sT) (i) "hard" sT followed by GNC and (ii) "soft" sT followed by GNC.

| HARD sT ACTIVATION | | SOFT sT ACTIVATION | |
|---|---|---|---|
| RA | BTI | RA | BTI |
| **61.47**±1.07 | -04.34±-1.34 | 60.86±0.37 | **-04.29**±-0.18 |

**Hard Sparse Coding Transform.** We define the "hard" sparse coding transform (sT) [6], as,

$$\text{sT}(\mathbf{z}_j^l) = \mathbf{z}_j^l \odot \mathbf{t}_j^l, \tag{9}$$

where $t_{j,i}^l = 1$ if $|z_{j,i}^l| \geq \lambda_j$, and $t_{j,i}^l = 0$ otherwise, while $\lambda_j = \frac{J}{N_l} \sum_{i=1}^{N_l/J} |z_{j,i}^l|$ is a sparsifying threshold. We have $\mathbf{z}_j^l$ as the linear activation's that belong to group $j \in \{1, 2, ..., J\}$, $\mathbf{z}^l = [(\mathbf{z}_1^l)^T, (\mathbf{z}_2^l)^T, ..., (\mathbf{z}_J^l)^T]^T \in \Re^{N_l}$.

**Soft Sparse Coding Transform.** The "soft" sparse coding transform (sT) is similar to the "hard" sparse coding transform, *i.e.*:

$$\text{sT}_{soft}(\mathbf{z}_j^l) = sign(\mathbf{z}_j^l) \odot \max(|\mathbf{z}_j^l| - \lambda_j, 0), \tag{10}$$

where we define the sparsifying threshold as above. The difference between (10) and (9) is that in (10) in addition to the tresholding to zero the magnitude of the nonzero values are also reduced by the sparsifying threshold $\lambda_j$.

Although (10) might have better bias/variance properties in classical signal processing applications, we empirically found out that (9) works better in our continual learning regimes. In Tab. 2, we verify this and give comparative results between "hard" and "soft" sT. We experimented using the CIFAR100 data set in the task-aware single pass regime. We set the reservoir size of 200 and use a reservoir batch size of 20. We use sT ("hard" or "soft") followed by GNC. As we can see in Tab. 2, the "hard" sT has an edge over "soft" sT, with higher retained accuracy of .6%, while the backward trasferability score is very similar between the "hard" sT and the "soft" sT.

## C. Additional Comparative Results with Normalization Methods

Table 3: Results using the CIFAR100 data set in the single pass continual learning regime. We show the performance when we use BN [4], GN [9], LN [2] and CN [5] at the penultimate layer. We experimented with ReLu before and after the normalization.

| | Linear Activation Followed by Normalization | | ReLu Activation Followed by Normalization | | Normalization Followed by ReLu | |
|---|---|---|---|---|---|---|
| NORM | RA | BTI | RA | BTI | RA | BTI |
| BN | **61.01**±.984 | -04.30±.319 | 56.90±2.05 | -06.80±0.52 | **60.65**±0.92 | -06.93±0.54 |
| GN | 59.47±.134 | -**02.20**±.026 | 56.49±0.50 | -04.50±.020 | 58.73±0.67 | -**04.66**±2.89 |
| LN | 60.46±3.76 | -02.46±.035 | **58.55**±0.58 | -05.53±.013 | 58.14±1.72 | -04.93±2.08 |
| CN | 60.18±.089 | -04.50±2.65 | 58.26±0.68 | -**04.50**±0.20 | 58.88±1.65 | -07.75±3.08 |

Table 4: Results using the CIFAR100 data set in the single pass continual learning regime. We show the performance when we different activation functions.

| | Activation Followed by GNC | | Activation Without GNC | |
|---|---|---|---|---|
| $f_n$ | RA | BTI | RA | BTI |
| LIN | 60.14±1.80 | -06.22±0.37 | **55.89**±0.38 | **-08.09**±1.06 |
| ReLU | 54.61±0.98 | -10.04±0.00 | 53.50±0.05 | -09.81±0.37 |
| sT$_{soft}$ | 60.86±0.37 | -04.29±-0.18 | 53.63±0.75 | -09.25±-0.61 |
| sT | **61.76**±2.03 | **-03.99**±1.09 | 54.74±0.80 | -09.18±0.03 |

In Tab. 3, we show the results in the single-pass regime using the CIFAR100 data set. We compare against existing normalization techniques, including including BN [4], GN [9], LN [2] and CN [5]. We used equivalent convolution neural network as in [3], and we apply the normalization at the penultimate layer. We use reservoir of size 200 and reservoir batch size of 20. In the GN, CN and GNC, we use 32 groups with 10 neuron activation's each for fair comparison.

As we can see in Tab. 3 on the left side on the top, when we do not apply non-linearity, BN, GN, LN and CN have good performance in both RA and BTI. While when we used ReLu activation followed by BN, GN, LN or CN we have lower retained accuracy (RA) and backward transferability (BTI). That is the retained accuracy over BN, GN, LN and CN drops by an average of around $3.5\%$. In the case when we preform the operations in the reversed order, that is BN, GN, LN or CN followed by ReLu the results have a bit higher retained accuracy compared to the previous ones. However, we still have lower retained accuracy compared to the linear activation by an average of around $2\%$. We also observed that when we used a sT activation followed by BN, GN, LN or CN, we had a strong degradation of performance (during training, the objective value got stuck at a certain value and was not able to improve/learn for the experimented normalization methods). We also tried out instance normalization (IN) [8] in a equivalent setup as before, but similarly as above, we were not able to make the model to learn.

In Tab. 4 on the bottom, we show the influence of the used nonlinear activation function before the GNC. As activation function we experimented with linear, ReLu, and sT. As reported in the main text, we can see that not all combinations of activation function followed by GNC are useful.

In Tab. 3 and Tab. 4, we can also see that our approach with "hard" sT and GNC has the best performance.

# D. Group-wise Neuron Competition Implementation

## D.1. Group-wise Neuron Competition (GNC) pythorch Implementation

In the following we prove a **pythorch** implementation of the group-wise normalization that induces group-wise neuron competition (GNC).

```
def gnc(a, group_size, num_of_groups):
    # GNC implementation
    # assuming shape_1 = group_size*num_of_groups
    shape_0, shape_1 = a.shape
    a = a.reshape(shape_0, group_size, num_of_groups)
    a = torch.nn.functional.normalize(a, p=2, dim=1).reshape(shape_0, shape_1)

    return  a
```

We note that before applying the GNC, we also apply "hard" sparse coding transform (sT) (for more details please see Appendix Section B.2.).

## D.2. Hard Sparse Coding Transform (sT) pythorch Implementation

In the following, we give the **pythorch** implementation for the "hard" sparse coding transform.

```
def st(z, group_size, num_of_groups):
    # sT implementation
    shape_0, shape_1 = z.shape
    # compute the threshold per group of neuron activation
    r = z.reshape(shape_0, group_size, num_of_groups ).detach().clone()
    tr = torch.abs( r ) >=  torch.mean( torch.abs(r), 1).unsqueeze(1)
    # set to zero the components below the threshold
    a = z * tr.float().reshape(shape_0, shape_1)

    return a
```

# References


[1] Abien Fred Agarap. Deep learning using rectified linear units (relu). *CoRR*, abs/1803.08375, 2018.

[2] Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *ArXiv*, abs/1607.06450, 2016.

[3] G. Gupta, Karmesh Yadav, and L. Paull. Look-ahead meta learning for continual learning. In *NeurIPS*, 2020.

[4] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, Proceedings of Machine Learning Research, pages 448–456. PMLR, 07–09 Jul 2015.

[5] Quang Pham, Chenghao Liu, and Steven HOI. Continual normalization: Rethinking batch normalization for online continual learning. In *International Conference on Learning Representations*, 2022.

[6] Saiprasad Ravishankar and Yoram Bresler. Sparsifying transform learning with efficient optimal updates and convergence guarantees. *IEEE Transactions on Signal Processing*, 63(9):2389–2404, may 2015.

[7] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986. Publisher: Nature Publishing Group.

[8] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022, 2016.

[9] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.