ECAI 2020

# Scalable deep learning: how far is one billion neurons?

## Optional assignment - understanding the benefits of *sparse training*

**Elena Mocanu and Decebal Mocanu [Lecturers][1] and Selima Curci [Instructor][2]**

*[1]Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, Enschede, 7522NB, the Netherland*
*[2] Department of Mathematics and Computer Science, Eindhoven University of Technology, 5600 MB Eindhoven, the Netherland*

## CIFAR10 classification task

Sparse neural networks are effective approaches to reduce the resource requirements for the deployment of deep neural network. A simple multi-layer perceptron trained with the Sparse Evolutionary Training (SET) algorithm introduced by [Mocanu *et al.* 2018] is able to substantially reduce the training time and outperform its dense counterpart as we could observe during the tutorial.

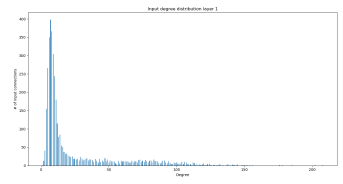
The advantages of shorter training time and memory usage are even more evident if we consider more complex datasets which have an higher number of features. CIFAR10 consists of various natural color images, each of which has $32X32$ pixels and belongs to one of ten classes. You can try to make the same comparisons as during the live session in terms of computational complexity and memory footprint by training SET-MLP, Dense-MLP (fully connected MLP) and FixProb-MLP (sparse MLP with fix topology) on this dataset (use the provided `monitor.py` file to log these metrics).

Hyperparameters for SET-MLP training with three hidden layers (4000, 1000, 4000) can be set to: sparsity level $\epsilon = 20$, rewire rate $\zeta = 0.3$, $dropout = 0.3$ and learning rate $\eta = 0.01$ with ReLu as activation function and cross entropy loss.

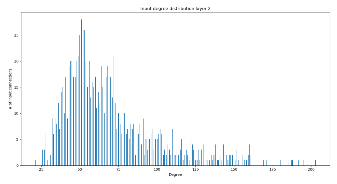
To achieve an higher accuracy and avoid overfitting I would suggest you to use the basic data augmentation strategy available in `keras` as reported in the following code snippet.

```
from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(
            featurewise_center=False,  # set input mean to 0 over the dataset
            samplewise_center=False,  # set each sample mean to 0
            featurewise_std_normalization=False,  # divide inputs by std of the dataset
            samplewise_std_normalization=False,  # divide each input by its std
            zca_whitening=False,  # apply ZCA whitening
            rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
            width_shift_range=0.1,  # randomly shift images horizontally (fraction of
                                              total width)
            height_shift_range=0.1,  # randomly shift images vertically (fraction of
                                              total height)
            horizontal_flip=True,  # randomly flip images
            vertical_flip=False)  # randomly flip images
datagen.fit(x)
output_generator = datagen.flow(x, y_true, batch_size=self.batch_size)
```

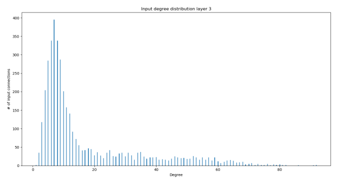
The original SET paper [Mocanu *et al.* 2018] shows that hidden neuron connectivity in SET-MLP rapidly evolve towards a power-law distribution. To understand better the effect of the topology evolution, try to inspect (plot) the degree distribution of the hidden neurons connections before training and at the end of the training process. You may expect to obtain something similar with Figure 1 which shows the input connection degree distribution of SET-MLP on CIFAR10 after 1000 epochs.



**(a)**



**(b)**



**(c)**

**Figure 1.** Input connections degree distribution after training SET-MLP for 1000 epochs with data augmentation on CIFAR10. (a) Layer 1; (b) layer 2; (c) layer 3.

## Input connectivity MNIST

Feature selection is a research topic, in which the objective is to determine the most important features of a dataset. It can be considered an optimization problem, in which we look for an optimal subset of features.

SET-MLP is able to find a topology where the input connections are focused on the most important feature of the input data. You can easily visualize this on the MNIST dataset. Observe how the input connectivity pattern changes over training and converges towarsd the most important pixels of the digits. You can save the input connectivity before the weight evolution step with the `get_core_input_connections` method of the `SET-MLP` class and run `plot_input_layer_connectivity.py` (you simply have to change the file path ) to obtain a nice animation of the input layer connectivity evolution during training. In the `"Pretrained_results"` folder there is a nice animation `"fashion_mnist_connections_evolution_per_input_pixel_rand0.gif"` for the FashionMNIST dataset (Figure 2).

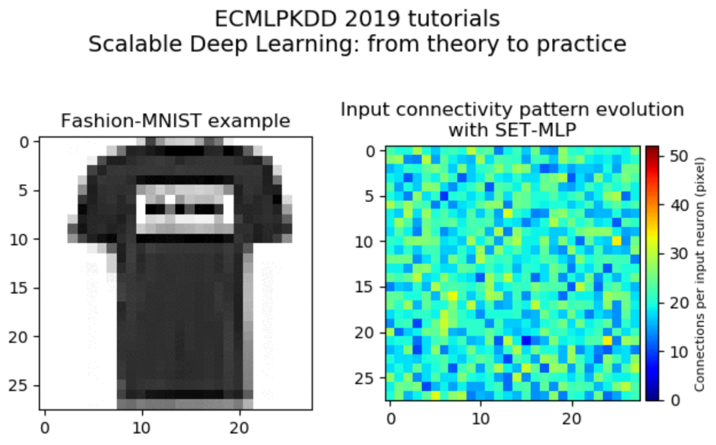


**Figure 2.** Input connectivity pattern evolution with SET-MLP on fashionmnist dataset over training.

The reference code for the tutorial is available here.

## Contact

For any question please feel free to contact us by email: s.curci@student.tue.nl, d.c.mocanu@utwente.nl.

## References

Mocanu, D. C., E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta. 2018. “Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science.” In *Nature Communications.*