Emergence of Latent Binary Encoding in Deep Neural Network Classifiers

Luigi Sbailò SBAILO@PHYSIK.HU-BERLIN.DE
Physics Department and IRIS Adlershof of the Humboldt-Universität zu Berlin, Berlin, Germany

Luca Ghiringhelli
Department of Materials Science and Engineering, Friedrich-Alexander Universität, Erlangen-Nürnberg, Germany
Physics Department and IRIS Adlershof of the Humboldt-Universität zu Berlin, Berlin, Germany

Editors: Sophia Sanborn, Christian Shewmake, Simone Azeglio, Nina Miolane

Abstract

We observe the emergence of binary encoding within the latent space of deep-neural-network classifiers. Such binary encoding is induced by introducing a linear penultimate layer, which is equipped during training with a loss function that grows as $\exp(\boldsymbol{x}^2)$, where $\boldsymbol{x}$ are the coordinates in the latent space. The phenomenon we describe represents a specific instance of a well-documented occurrence known as *neural collapse*, which arises in the terminal phase of training and entails the collapse of latent class means to the vertices of a simplex equiangular tight frame (ETF). We show that binary encoding accelerates convergence toward the simplex ETF and enhances classification accuracy.

Keywords: Binary encoding, neural collapse, latent space.

1. Introduction

In tasks like images classification, deep neural networks have achieved levels of performance that surpass human capabilities. Nevertheless, there is a general lack of understanding regarding the theoretical mechanisms behind these outstanding results.

In the last years, there has been a growing interest in studying the geometrical structures that emerge in the latent space of deep neural networks. Notably, it has been noticed that the class means in the penultimate layers collapse to the vertices of a simplex equiangular tight frame (ETF) in the terminal phase of training Vardan Papyan (2020). This phenomenon known as *neural collapse* has been linked to transfer learning Galanti et al. (2022) and incremental learning Yibo Yang (2023).

In this work, we develop a method for generating a binary encoding for the latent representations found in the penultimate layer of deep neural networks. Within each dimension of the penultimate layer, latent representations are in practice trained to assume one of two possible values. In the tests that we perform all data points belonging to the same class adopt an identical binary encoding. This implies that data points within the same class ultimately converge to the vertices of a simplex constructed on the vertexes of a binary hypercube. Notably, our work represents a specific instance of *neural collapse*. In our case, the phenomenon extends beyond just the class means collapsing to simplex vertices; rather, it encompasses all data points within the same class. Furthermore, our findings demonstrate that this method enhances accuracy of neural-networks prediction.

2. Binary Encoding

Given a labeled dataset $\{\boldsymbol{x}, \overline{\boldsymbol{y}}\}$, we deal with the problem of predicting the labels with a deep neural network. For an input point $\boldsymbol{x}$, we can break down the process of producing the neural network's output $\boldsymbol{f}(\boldsymbol{x})$ into two distinct steps. Firstly, the non-linear component of the neural network generates a latent representation $\boldsymbol{h}(\boldsymbol{x})$. Following this, a linear classifier, characterized by weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$ operates on this latent representation to compute $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{b}$. The predicted label $\boldsymbol{y}$ is finally determined by applying a softmax function to the network's output. The neural network is trained through the minimization of the cross-entropy loss function $\mathcal{L}_{\mathrm{CE}}\left(\boldsymbol{f}(\boldsymbol{x}), \overline{\boldsymbol{y}}\right)$, which measures the disparity between the network's predictions and the ground truth labels.

Here, we propose the introduction of an additional linear layer prior to the classifier, defined as $\boldsymbol{x}_{bin} = \boldsymbol{W}_{bin}\boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{b}_{bin}$. This layer serves as the penultimate step in the network architecture, with classification being subsequently determined through another linear operation, $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x}_{bin} + \boldsymbol{b}$. In addition to the cross-entropy loss applied to the network's output, we incorporate a loss function into the penultimate layer defined as: $\mathcal{L}_{\mathrm{Bin}}(\mathbf{x}_{\mathrm{bin}}) = e^{\boldsymbol{x}^2_{\mathrm{bin}}}$. The resulting loss function is thus composed of two terms:

$$\mathcal{L} = \mathcal{L}_{\mathrm{CE}} + \gamma\, \mathcal{L}_{\mathrm{Bin}}, \tag{1}$$

where γ is a hyperparameter.

The latent binary encoding emerges as a result of balancing two conflicting tendencies coming from the two components of the loss function. The exponential loss function pushes towards having all latent representations closer to zero, while the minimization of the cross-entropy induces differentiation among the different latent representations. This dynamics naturally leads to a configuration where, for each latent dimension, most of the latent values cluster around two opposing peaks in relation to the origin. The presence of these two peaks facilitates the necessary differentiation for distinguishing various representations, and they tend to be in close proximity to zero due to the influence of the exponential loss. Assuming the concentrated distribution of points only around the two peaks, it becomes evident that the latent representation effectively approximates only two distinct values: either positive or negative.

3. Experiments

To evaluate the impact of incorporating a binary encoding layer, we conducted experiments with four distinct neural network architectures, all of which share a common base network responsible for generating the latent representation $\boldsymbol{h}(\boldsymbol{x})$. Nevertheless, these networks diverge in their subsequent steps for classification.

One network architecture, referred as *Binary encoding*, implements a linear penultimate layer with loss function as in Eq (1). The *Linear penultimate* architecture features a linear penultimate layer as well, but is trained using only the cross-entropy loss function. The *Non-linear penultimate* architecture implements a non-linear layer that acts on $\boldsymbol{h}(\boldsymbol{x})$ before linear classification. The fourth *No penultimate* architecture performs linear classification directly on the $\boldsymbol{h}(\boldsymbol{x})$ latent representation.

We note that the *Binary encoding*, *Linear penultimate* and *Non-linear penultimate* architectures have the same number of layers and parameters but differ for activation and loss functions, while the *No penultimate* architecture has one layer less with respect to the others. These 4 different architectures are tested on MNIST and FashionMNIST. Details about training and architecture of the network used to generate the latent representation $\boldsymbol{h}(\boldsymbol{x})$ are given in Appendix A. Code to reproduce results presented in this work is available online[1].

Figure 1: Average log-likelihood scores and standard deviations for Gaussian mixture models with two modes on each dimension of the penultimate layer. Averages of different training outcomes are shown, and line shadows represent the standard deviations. If not visible standard deviations are small for the image resolution.

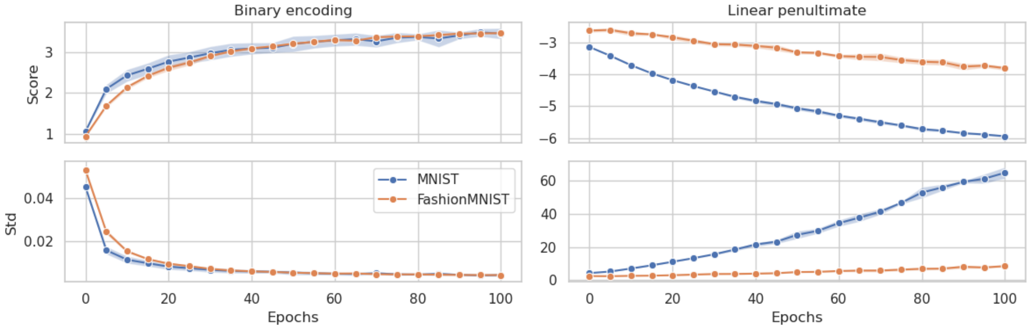
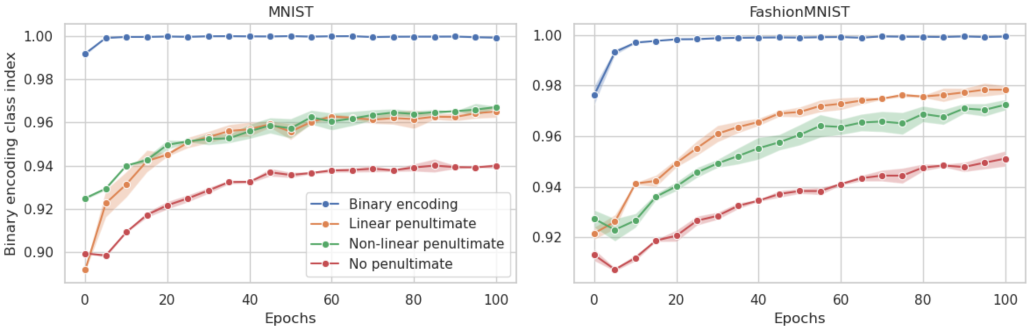


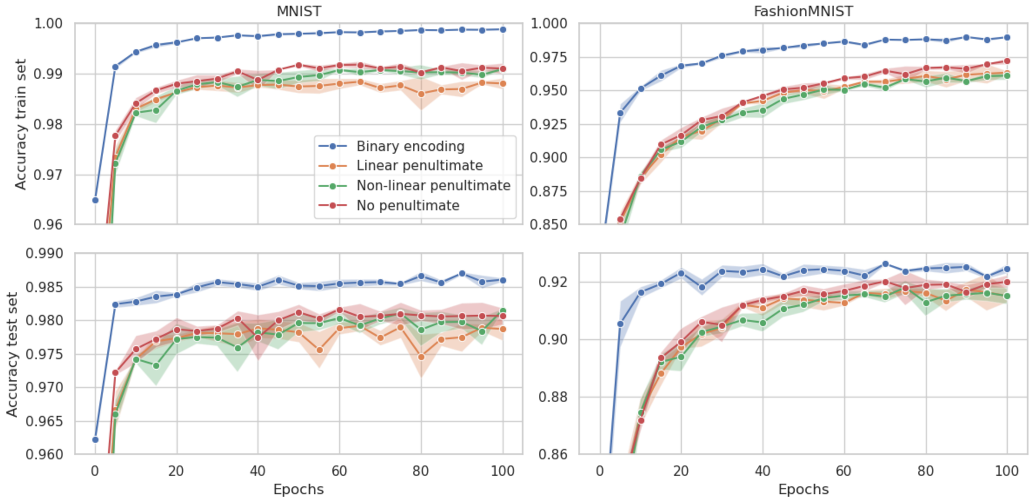
Figure 2: Fraction of points belonging to the same class which share the same binary encoding. Averages of different training outcomes are shown, and line shadows represent the standard deviations. If not visible standard deviations are small.



In order to test the binarity hypothesis, which we define as the assumption that each dimension in the latent representation can assume approximately only one of two values, we

[1] https://github.com/luigisbailo/emergence_binary_encoding.git

Figure 3: Neural network accuracy on train and test set. Averages of different training outcomes are shown, and line shadows represent the standard deviations.



fit a Gaussian mixture model with 2 modes on the *Binary encoding* latent representation $\mathbf{x}_{\text{Bin}}$. A different fit is performed on each dimension over all values of the training set. In each dimension, the average log-likelihood score of the training set is computed and averaged over all dimensions. Also the standard deviation of the two posterior distributions are collected and averaged over all dimensions. These values are plotted in Fig. 1, where we can see that during training the score increases while the standard deviation decreases. This observation supports our binarity hypothesis, as it aligns with the notion that a Gaussian distribution with a standard deviation approaching zero implies that all data points converge to a single position. The same analysis is performed for the *Linear penultimate* architecture as it also features a linear layer before classification. However, we can see that in this architecture the binarity hypothesis does not hold.

As we assume that each latent representation $\boldsymbol{x}_{\text{Bin}}$ is encoded into a binary representation, we expect that points with the same label present the same encoding. We then generate a binary encoding of the penultimate layer in each of the network architectures we study. This encoding is generated giving 1 to all positive values and 0 to values equal to or lower than 0. In Fig. 2, we show the fraction of points that share the same encoding and belong to the same class. Notably, we observe that this assumption holds true exclusively for the *Binary encoding* architecture. All points belonging to the same class are in fact placed on a vertex of the simplex designed on the vertices of a hypercube. Binary encoding also accelerates *neural collapse* as discussed in Appendix B.

Finally, we can see in Fig. 3 that the implementation of a *binary encoding layer* has the effect of improving the accuracy of neural network classification both on the train and test set, while the other three architectures show comparable performance.

4. Conclusion and limitations

We have discussed a method to generate a binary encoding in the latent space, which is accomplished by adding a penultimate *binary encoding layer*, i.e. a linear layer that incorporates an exponentially growing loss function. The emergence of this phenomenon is shown to accelerate convergence toward the vertices of a simplex equiangular tight frame, and to enhance the network accuracy. Although results seem to be promising to suggest that binary encoding should be used to enhance network performance, more comprehensive tests on more complex datasets and with more expressive deep neural networks are still to be done.

Appendix A. Training and architecture details

To generate the latent representation $\boldsymbol{h}(\boldsymbol{x})$, two distinct neural network architectures were employed for the two different datasets. For MNIST classification, a fully connected neural network with three layers, each consisting of 2048 nodes, along with two dropout layers, was utilized. In the case of FashionMNIST, a convolutional neural network was employed, featuring five convolutional layers with specified input and output channel configurations ([1,64],[64,128],[128,256],[256,256], [256,512]), a kernel size of 3, and padding of 1. Additionally, three max pool layers with a kernel size of 2 and a stride of 2 were applied in the following sequence: Conv2D, MaxPool, Conv2D, MaxPool, Conv2D, Conv2D, MaxPool, Conv2D. Furthermore, two fully connected layers, each comprising 1024 nodes, were employed after the convolutional layers, and two dropout layers were included. A nonlinear activation function was consistently applied following each convolutional layer.

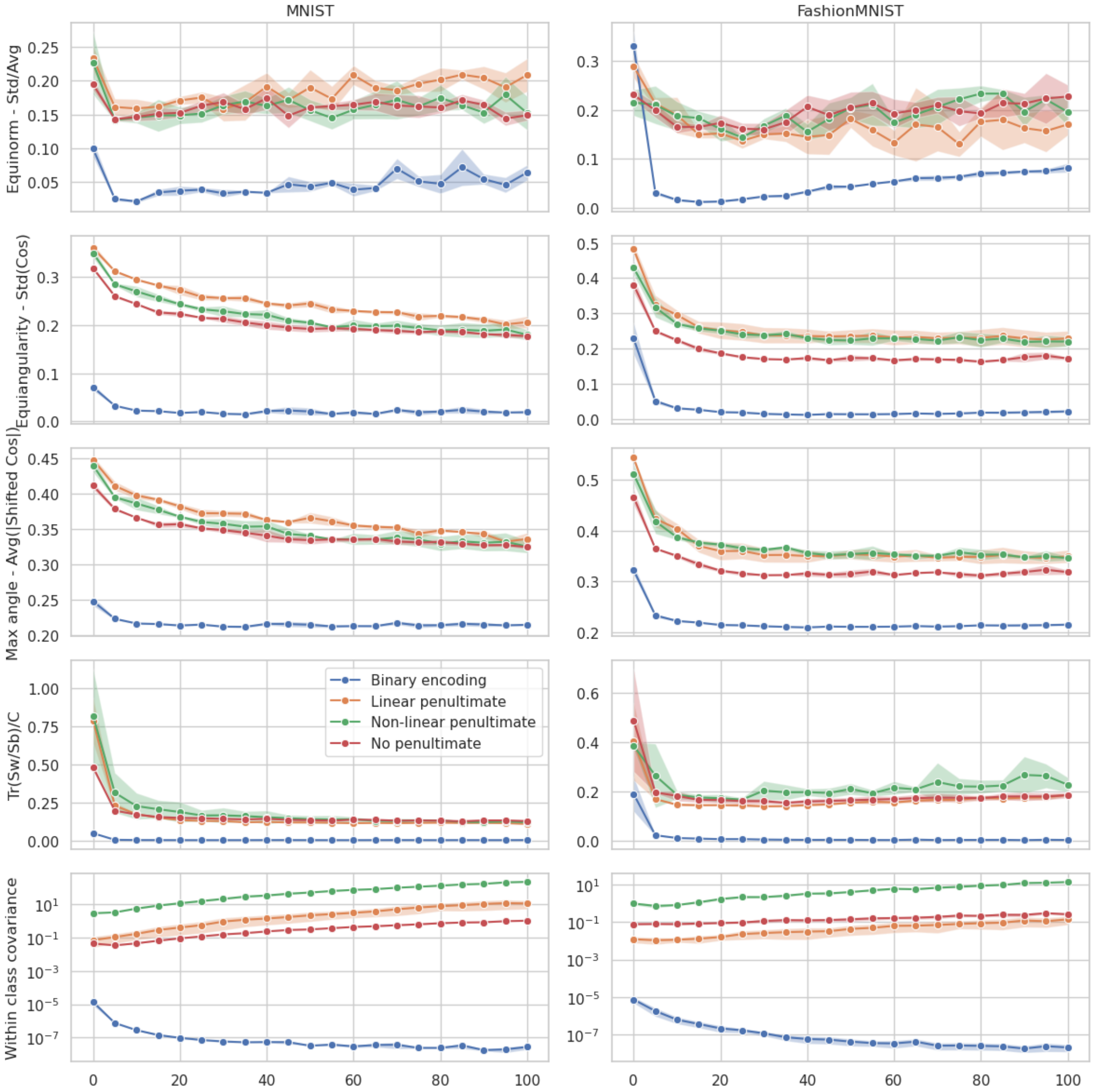
Both architectures incorporated a dropout rate of 0.5 and utilized ReLU activation functions. In the context of the CNN network applied to FashionMNIST, the *Binary encoding*, *Linear penultimate*, and *Non-linear penultimate* architectures included an additional fully connected layer consisting of 128 nodes. Conversely, for the fully connected network applied to MNIST, this penultimate layer contained 64 nodes. The *Non-linear penultimate* architecture incorporated a ReLU activation function within this penultimate layer.

The training process utilized the Adam optimizer with default settings as specified in the PyTorch implementation, employing a learning rate of 10^{-4}. The learning rate was reduced by half after every 20 epochs. For the MNIST dataset, a batch size of 64 was employed, whereas for the FashionMNIST dataset, a batch size of 128 was utilized. The loss function employed for the 'Binary encoding' architecture was as in Eq. (1) with $\gamma = 10$. Each network architecture has undergone training three times with distinct initial conditions, and the quantities displayed in plots represent the averages and standard deviations of these outcomes.

Appendix B. Convergence to simplex equiangular tight frame

In the terminal phase of training the vectors of the class means are known to converge to a simplex equiangular tight frame (ETF) as manifestation of a phenomenon known as *neural collapse*. The class mean vectors converge to having equal lengths, resulting in uniform angles between any given pair of vectors. The ETF configuration represents the maximum pairwise distance while adhering to the aforementioned properties.

Figure 4: Plots demonstrating convergence to the vertices of a simplex equiangular tight frame. From top to bottom: 'Equinorm' as variation of the mean classes norms; 'Equiangularity' as variation of the angle between all class means pairs; 'Max Angle' as distance from the max angle class means can have; 'Tr(Sw/Sb)/C' as weigheted within-class variance; 'Within class covariance' is the average of the within-class covariance matrix. More details about the quantities plotted can be found in Ref. Vardan Papyan (2020).



In Figure 4, we illustrate this convergence of class means towards the vertexes of the simplex ETF. Notably, we observe that this convergence occurs more rapidly when employing a *binary encoding layer*. In this figure the within class variation is computed using the within class covariance compared with the between class covariance. The collapse of variability becomes apparent when contrasting with the between-class covariance. However, in the lower plot, which displays the average within-class covariance matrix, we observe an interesting trend. The average within-class covariance matrix itself, without comparison to the between-class covariance matrix, shows an increase during training, unless the *binary encoding layer* is integrated into the network. This implies that, when using the *binary encoding layer*, not only the class means but also all data points within the dataset converge towards the vertexes of a simplex ETF.

References

Tomer Galanti, András György, and Marcus Hutter. On the role of neural collapse in transfer learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=SwIp410B6aQ.

David L. Donoho Vardan Papyan, X. Y. Han. Prevalence of neural collapse during the terminal phase of deep learning training. *PNAS*, 117:24652–24663, 2020.

Xiangtai Li Zhouchen Lin Philip Torr Dacheng Tao Yibo Yang, Haobo Yuan. Neural collapse inspired feature-classifier alignment for few-shot class incremental learning. *arXiv:2302.03004*, 2023.