Indirectly Parameterized Concrete Autoencoders

Alfred Nilsson [*1,2] **Klas Wijk** [*1,3] **Sai bharath chandra Gutha** [1] **Erik Englesson** [1] **Alexandra Hotti** [1,2,4]
Carlo Saccardi [1] **Oskar Kviman** [1,2] **Jens Lagergren** [1,2] **Ricardo Vinuesa** [1,3] **Hossein Azizpour** [1,3]

Abstract

Feature selection is a crucial task in settings where data is high-dimensional or acquiring the full set of features is costly. Recent developments in neural network-based embedded feature selection show promising results across a wide range of applications. Concrete Autoencoders (CAEs), considered state-of-the-art in embedded feature selection, may struggle to achieve stable joint optimization, hurting their training time and generalization. In this work, we identify that this instability is correlated with the CAE learning duplicate selections. To remedy this, we propose a simple and effective improvement: Indirectly Parameterized CAEs (IP-CAEs). IP-CAEs learn an embedding and a mapping from it to the Gumbel-Softmax distributions' parameters. Despite being simple to implement, IP-CAE exhibits significant and consistent improvements over CAE in both generalization and training time across several datasets for reconstruction and classification. Unlike CAE, IP-CAE effectively leverages non-linear relationships and does not require retraining the jointly optimized decoder. Furthermore, our approach is, in principle, generalizable to Gumbel-Softmax distributions beyond feature selection.

1. Introduction

Feature selection is a fundamental task in machine learning and statistics, enabling more parsimonious and interpretable models. It is essential in several applications such as bioinformatics e.g., gene subset selection, neuroscience e.g., fMRI analysis, and fluid mechanics e.g., optimal sensor placement. Moreover, feature selection is often used for regularization. Unfortunately, finding the optimal selection is NP-hard (Amaldi & Kann, 1998).

Although a large body of work exists on feature selection (Cai et al., 2018), due to the success of deep networks, neural network-based embedded feature selection has gained more interest (Balın et al., 2019; Yamada et al., 2020; Lemhadri et al., 2021). Among those, Concrete Autoencoders (CAEs) (Balın et al., 2019), is an established approach which allows for differentiable feature selection using a layer consisting of stochastic Gumbel-Softmax distributed nodes (Maddison et al., 2017; Jang et al., 2017).

In this work, we identify a recurring instability issue of CAEs (Figure 1, top) which leads to increased training time and subpar performance. We then show that the instability correlates with selection of redundant features (Figure 1, bottom). To remedy this, we propose a simple modification to indirectly parametrize the Gumbel-Softmax distributions via a learnable embedding and transformation (Figure 2), we refer to this alternative as Indirectly Parameterized CAEs (IP-CAEs) and rigorously verify their empirical effectiveness. We summarize this paper's main contributions below.

- We identify training instability in CAE and show it strongly correlates with redundant features (Figure 1).
- We introduce IP-CAE (Figure 2), a simple and effective way to alleviate the instability of vanilla CAE (Figure 5a solid lines), and show it leads to unique selections (Figure 5a dotted lines), improved accuracy (Figure 5b) and training time (Table 3). We also study the update rules of CAE and IP-CAE, showing the latter learns a transformation of gradients (Section 2.4).
- We propose and compare against Generalized Jensen-Shannon Divergence (GJSD) regularization (Section 2.5) , an explicit, probabilistic approach to mitigate duplicate selections, and show while GJSD regularization is effective, IP-CAE is superior (Tables 1 and 2).
- We demonstrate successful end-to-end training of CAE architectures for both reconstruction (Table 1) and classification (Table 2) achieving state-of-the-art results on multiple datasets.

[*]Equal contribution [1]KTH Royal Institute of Technology, Stockholm, Sweden [2]Science for Life Laboratory, Solna, Sweden [3]Swedish e-Science Research Centre (SeRC), Stockholm, Sweden [4]Klarna, Stockholm, Sweden. Correspondence to: Alfred Nilsson <alfredn@kth.se>, Klas Wijk <kwijk@kth.se>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

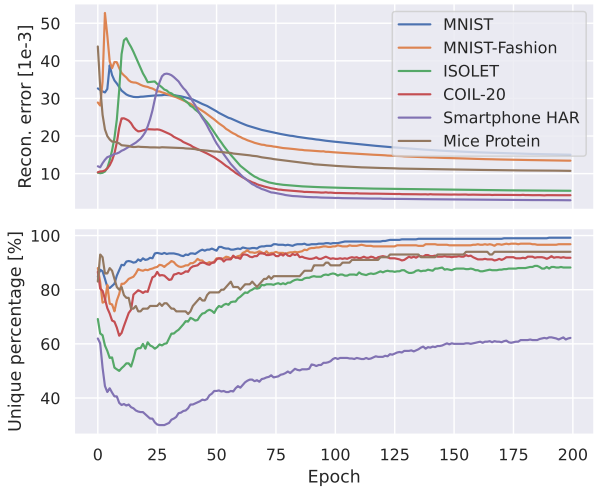
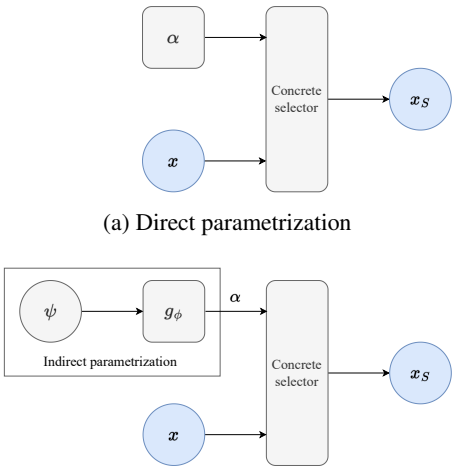


Figure 1: **CAE Training Instability**. For most datasets, the CAE architecture exhibits a large spike in reconstruction error that consistently correlates with the unique percentage (definition 2.1).



(a) Direct parametrization

(b) Indirect parametrization

Figure 2: **Architecture**. An overview of the CAE architecture, showcasing Indirect Parametrization (IP). Instead of directly learning α, we propose to learn an embedding ψ and a transformation g_ϕ that output α.

- We study the various aspect of IP-CAE and specifically show that it does not require additional hyperparameter tuning and that its superior performance is insensitive to the number of selected features (Figure 7) and the size of indirect parametrization (Figure 4).

2. Method

In this section we describe the vanilla CAE, discuss its shortcomings and present our proposed improvements to CAE training. We introduce a new way to *indirectly* parameterize the Gumbel-Softmax distributions in CAE, which we call IP. We further propose a baseline diversity-encouraging regularization method using the Generalized Jensen Shannon Divergence (GJSD).

2.1. Concrete Autoencoders (CAE)

The CAE (Balın et al., 2019) architecture is competitive with state-of-the-art methods in neural network-based embedded feature selection. CAEs consist of two components: a concrete selection layer that performs differentiable feature selection on the input features (encoder) and an arbitrary neural network (decoder). A predictive network is used in place of the decoder for classification or regression tasks.

The concrete selector layer consists of K independent Gumbel-Softmax distributed variables $\boldsymbol{m}_j$ (Jang et al., 2017; Maddison et al., 2017):

$$\boldsymbol{m}_j = \frac{\exp\{(\log \boldsymbol{\alpha}_j + \boldsymbol{g}_j)/T\}}{\sum_{i=1}^{D} \exp\{(\log \boldsymbol{\alpha}_{j,i} + \boldsymbol{g}_{j,i})/T\}}, \tag{1}$$

where $\log \boldsymbol{\alpha}_j \in \mathbb{R}^D$ for $j \in 1, 2, \ldots, K$ are the distributions parameters (logits), $\boldsymbol{g}_j \in \mathbb{R}^D$ are i.i.d. standard Gumbel distributed (Gumbel, 1954), and $T \in \mathbb{R}_+$ is a global temperature that is annealed throughout the training.

The samples are multiplied with the input features and passed through the decoder network. Using the reparametrization trick, the parameters are learnable through backpropagation from the decoder network's output. By forming a matrix whose rows contain $\{\mathbf{m}_j\}_{j=1}^k$ and denoting it by $\boldsymbol{M} \in \mathbb{R}^{K \times D}$, we can express the complete subset selection according to CAE as:

$$\mathbf{x}_S = \boldsymbol{M}\mathbf{x}, \tag{2}$$

where $\mathbf{x}_S \in \mathbb{R}^K$. Then, the selected features serve as the input to an arbitrary neural network f_θ the output of which is used to calculate a loss. MSE and cross-entropy losses are commonly used for reconstruction and classification respectively.

Balın et al. (2019) propose exponential annealing from a starting temperature T_0 to a final temperature T_B according to the following annealing schedule which we also use:

$$T(b) = T_0 \left(\frac{T_B}{T_0}\right)^{\frac{b}{B}}, \tag{3}$$

where $b \in \mathbb{N}$ is the current epoch and $B \in \mathbb{N}$ is the total number of epochs. The authors find that this schedule works for a broad range of datasets and is not sensitive to the specific start and end temperatures chosen.

As $T \to 0$, the Gumbel-Softmax samples $\boldsymbol{m}_j$ approach one-hot vectors corresponding to single input features. At

test time, we evaluate the decoder using *discrete* input features selected according to $\arg\max_j \log \boldsymbol{\alpha}_{i,j}$, where $i \in \{1, 2, \ldots, K\} = [K]$.

2.2. Challenges of CAE

In principle, it is not guaranteed that the learned parameters of the Gumbel-Softmax will correspond to distinct input features at any given point during training. We quantify the diversity of the Gumbel-Softmax parameters using the Unique Percentage (UP).

Definition 2.1 (Unique Percentage)**.** For a given set of Gumbel-Softmax parameters (logits) $\log \boldsymbol{\alpha} \in \mathbb{R}^{K \times D}$:

$$UP(\boldsymbol{\alpha}) = 100 \cdot \frac{|\{\arg\max_j \log \boldsymbol{\alpha}_{i,j} : i \in [K]\}|}{K}, \quad (4)$$

is the percentage of unique maximum parameter indices. Note that D denotes the total number of features and K the number of selected features.

We empirically demonstrate instability during training of CAEs (Figure 1, top). Interestingly, our results show that this instability strongly correlates with the unique percentage, consistently across tasks and datasets (Figure 1, bottom).

Furthermore, we empirically establish three more shortcomings of CAEs: (i) a large number of training epochs is required for CAE to converge to a local minimum, (ii) the quality of these local minima sometimes exhibit a high variance, and (iii) an end-to-end optimization of a non-linear decoder might incur additional instability, especially for prediction tasks other than reconstruction.

In the next subsection we describe our proposed IP-CAE which is later shown to alleviate the aforementioned shortcoming of vanilla CAE including instability, unique percentange, variance (Figure 5), training time (Table 3), and non-linear decoder (Figure 6) leading to state-of-the-art performance for feature selection for both reconstruction (Table 1) and classification (Table 2) on all datasets considered.

2.3. Indirect Parametrization

We investigate parameterizing $\log \boldsymbol{\alpha} \in \mathbb{R}^{K \times D}$ by transforming an array of learnable parameters $\boldsymbol{\Psi} \in \mathbb{R}^{K \times P}$ with a network g_ϕ as follows:

$$\log \boldsymbol{\alpha}_i = g_\phi(\boldsymbol{\psi}_i), \quad (5)$$

where $\log \boldsymbol{\alpha}_i \in \mathbb{R}^D$ and $\boldsymbol{\psi}_i \in \mathbb{R}^P$ are the transposed ith rows of $\log \boldsymbol{\alpha}$ and Ψ, respectively. In our experiments, we let g_ϕ be a linear network that is shared across stochastic nodes, *i.e.* $\phi = (\boldsymbol{W}, \boldsymbol{b})$ and:

$$\log \boldsymbol{\alpha}_i = \boldsymbol{W}\boldsymbol{\psi}_i + \boldsymbol{b}, \quad i \in [K], \quad (6)$$

where $\boldsymbol{W} \in \mathbb{R}^{D \times P}$ and $\boldsymbol{b} \in \mathbb{R}^D$. This can be interpreted as a feature embedding with embedding dimensionality P.

2.4. A Comparison Between CAE and IP-CAE

As the difference between CAE and IP-CAE is in how $\log \boldsymbol{\alpha}$ is parameterized, we analyze the differences between the methods by studying the following update rule based on different parameterizations: $\log \boldsymbol{\alpha}_i^{(t+1)} \leftarrow \log \boldsymbol{\alpha}_i^{(t)} - \eta \nabla \mathcal{L}$, where $\mathcal{L}$ is the gradient of the loss with respect to $\log \boldsymbol{\alpha}_i^{(t)}$.

CAE can be interpreted as a trivial case of IP, where $\log \boldsymbol{\alpha}$ is directly parameterized by a learnable buffer $\boldsymbol{\Psi} \in \mathbb{R}^{K \times P}$ with $P = D$, such that $\log \boldsymbol{\alpha}_i = \boldsymbol{\psi}_i$. In this case, the update rule for $\log \boldsymbol{\alpha}_i$ is:

$$\log \boldsymbol{\alpha}_i^{(t+1)} \leftarrow \boldsymbol{\psi}_i - \eta \nabla \mathcal{L} \quad (7)$$

where t is the current optimization step, η is the learning rate, and $\nabla \mathcal{L} \in \mathbb{R}^D$ is the gradient of the loss function $\mathcal{L}$ with respect to $\log \boldsymbol{\alpha}_i^{(t)} = \boldsymbol{\psi}_i^{(t)} = \boldsymbol{\psi}_i$.

For simplicity, we consider IP-CAE with $P = D$ and without the bias term, and thus have $\log \boldsymbol{\alpha}_i = \boldsymbol{W}\boldsymbol{\psi}_i$, with the following update rule (detailed derivations in Appendix D):

$$\log \boldsymbol{\alpha}_i^{(t+1)} \leftarrow \boldsymbol{W}\boldsymbol{\psi}_i - \eta \boldsymbol{T}_i \nabla \mathcal{L} \quad (8)$$

$$\boldsymbol{T}_i = \boldsymbol{W}\boldsymbol{W}^T + \boldsymbol{\psi}_i^T(\boldsymbol{\psi}_i - \eta \boldsymbol{W}^T \nabla \mathcal{L})\boldsymbol{I} \quad (9)$$

where $\nabla \mathcal{L}$ is the gradient of the loss with respect to $\log \boldsymbol{\alpha}_i^{(t)} = \boldsymbol{W}^{(t)}\boldsymbol{\psi}_i^{(t)} = \boldsymbol{W}\boldsymbol{\psi}_i$, and the step-dependent matrix $\boldsymbol{T}_i \in \mathbb{R}^{D \times D}$ represents a learned transformation of the gradients. The transform affects the gradients in two ways: a linear transformation represented by $\boldsymbol{W}\boldsymbol{W}^T$ that is shared for all i, and a scaling by $\boldsymbol{\psi}_i^T(\boldsymbol{\psi}_i - \eta \boldsymbol{W}^T \nabla \mathcal{L}) \in \mathbb{R}$, which is the dot product between $\boldsymbol{\psi}_i^{(t)}$ and $\boldsymbol{\psi}_i^{(t+1)}$. A geometric interpretation of the dot product is $||\boldsymbol{\psi}_i^{(t)}||_2 ||\boldsymbol{\psi}_i^{(t+1)}||_2 \cos(\theta_i)$ with θ_i being the angle between the two vectors. Empirically, we have found that the learned rescaling changes throughout training, and our results suggest that these changes are beneficial. Because of the interactions between $\boldsymbol{W}$, ϕ, and $\nabla \mathcal{L}$, the effect throughout training may be elaborate. We include additional experiments exploring the update's behavior in Appendix C.

Thus, simply changing the parameterization of $\log \boldsymbol{\alpha}_i$ from $\boldsymbol{\psi}_i$ to $\boldsymbol{W}\boldsymbol{\psi}_i$ significantly changes the update rule to transform all gradients by $\boldsymbol{W}\boldsymbol{W}^T$ and scale specific gradients by the dot product between the current and next step of $\boldsymbol{\psi}_i$. Clearly, IP-CAE is a generalization of CAE, as CAE corresponds to the special case with a fixed $\boldsymbol{W} = \boldsymbol{I}$.

2.5. Generalized Jensen-Shannon Divergence

Since the aforementioned challenges of CAE (Section 2.2), particularly its instability, is strongly correlated with reduced unique percentage (Figure 1), we propose a direct

mechanism to encourage diversity of selected features by using the GJSD as a regularization, which will serve as an important baseline in our empirical study. We later show its effectiveness in comparison to the vanilla CAE but IP remains superior (Tables 1 and 2).

Definition 2.2 (Generalized Jensen–Shannon Div.)**.** The GJSD for K categorical distributions $\{\boldsymbol{p}_i\}_{i=1}^K$, and weights $\mathbf{w}$ is given by:

$$D_{GJS}(\{\boldsymbol{p}_i\}_{i=1}^K) = \sum_{i=1}^K w_i D_{KL}\left(\boldsymbol{p}_i \,||\, \sum_{j=1}^K w_j \boldsymbol{p}_j\right), \quad (10)$$

where D_{KL} denotes the Kullback–Leibler divergence.

GJSD has previously been employed to measure the diversity among the mixture components (Kviman et al., 2022; 2023) and that can be utilized as a loss function (Englesson & Azizpour, 2021; Hendrycks et al., 2019). As the goal is to learn distinct Gumbel-Softmax distributions that converge to unique features, maximizing the D_{GJS} can help prevent duplicate selections by encouraging diverse distributions. While we are concerned with Gumbel-Softmax distributions, they can be approximately treated as categoricals. We exploit this by calculating an approximate GJSD with the probability vectors $S(\log \boldsymbol{\alpha}_i)$ of our Gumbel-Softmax distributions, where $S(\boldsymbol{z}) = \exp\{[z_1, \ldots, z_K]\} / \sum_{i=1}^K \exp\{z_i\}$ is the softmax function. We assume equal weights for the mixture components, *i.e.* $w_i \equiv 1/K$.

Using definition 2.2, we define our regularized loss function

$$\mathcal{L}_\lambda(\cdot, \log \boldsymbol{\alpha}) = \mathcal{L}(\cdot) - \lambda D_{GJS}(\{S(\log \boldsymbol{\alpha}_i)\}_{i=1}^K), \quad (11)$$

where $\lambda > 0$ is a parameter controlling the regularization and $\mathcal{L}(\cdot)$ is the non-regularized loss (e.g. MSE for reconstruction and cross-entropy for classification).

3. Related Work

Feature selection methods are broadly categorized into three paradigms: filter methods, wrapper methods, and embedded methods (Guyon & Elisseeff, 2003). While filter methods treat each feature independently and do not account for interactions between them, wrapper methods select features based on a black-box model. Finally, embedded methods perform feature selection as part of the model, usually through learning the feature selection throughout fitting the model.

Next we describe recent state-of-the-art methods used for neural network-based embedded feature selection.

Feature selection using STGs Yamada et al. (2020), similarly to CAE, use stochastic nodes to sample features during training to perform differentiable joint optimization of feature selection and model. While CAE models the selection using K categorical nodes in the concrete selector layer, Stochastic Gaussian Gates (STGs) uses D Bernoulli nodes, each for one input feature, where K denotes the desired number of optimal features, and D denotes the total number of input features. The authors propose a novel reparametrization for Bernoulli variables using thresholded Gaussian variables to allow for differentiable learning. The use of Bernoulli gates is closely related to the Bernoulli-Gaussian model for linear regression with feature selection and ℓ_0 regularization.

LassoNet Lemhadri et al. (2021) extend the popular Lasso (Tibshirani, 1996) method for regression. Although the classic Lasso has been efficient and useful for embedded feature selection in linear models, it is challenging to generalize it to neural network models (Cui & Wang, 2016). The LassoNet architecture achieves this by introducing residual connections from the input layer to the output of the network and applying an ℓ_1 penalty term to that layer. The design principle is such that it allows for a feature to be selected by the model if and only if it also gets selected by the residual layer. The authors model this principle as an explicit constraint in the optimization problem and propose a projected proximal gradient optimization algorithm to ensure the constraint satisfiability during the process.

In Section 4 we demonstrate state-of-the-art performance of IP-CAE compared to the methods listed above.

Other feature selection methods Deep Lasso (Cherepanova et al., 2023) is another generalization of the Lasso, different from LassoNet. Instead of penalizing the weights directly, the authors suggest penalizing the gradient and recovering the classic Lasso as a special case of their method. Another approach is training sparse neural networks where the sparse input layer naturally performs feature selection (Louizos et al., 2018; Sokar et al., 2022). Although not trivially extended to neural networks, sparse priors formalize and extend the Lasso approach (Carvalho et al., 2009; Ročková & George, 2018). Deep Knockoffs (Romano et al., 2020) use deep generative models to enhance knockoff machines (Barber & Candès, 2015), a powerful method for statistical variable selection. Selecting measurements in compressed sensing, an important problem in medical imaging, has seen the development of similar methods to those in feature selection (Bakker et al., 2020; Huijben et al., 2019). Finally, instance-wise feature selection extends the problem of global feature selection to predict selections per sample, providing a hard, thresholded explanation of the network's prediction (Yoon et al., 2018; Chen et al., 2018).

End-to-end learnable EEG channel selection Strypsteen & Bertrand (2021) focus on a unified approach for select-

Table 1: **Reconstruction Error**. Mean normalized Frobenius norm for the reconstruction task. The values are an average of 10 repetitions ± 1 standard deviation.

MODEL	MNIST	MNIST-FASHION	ISOLET	COIL-20	SMARTPHONE HAR	MICE PROTEIN
STG	2.42E-02 ± 2.70E-06	1.80E-02 ± 2.84E-06	7.00E-03 ± 1.96E-06	5.00E-03 ± 6.75E-06	4.00E-03 ± 3.04E-06	1.17E-02 ± 6.43E-05
LASSONET	1.98E-02 ± 9.02E-06	1.96E-02 ± 4.23E-06	7.40E-03 ± 4.25E-06	6.24E-03 ± 5.42E-06	4.01E-03 ± 1.29E-06	1.16E-02 ± 4.45E-05
CAE	1.48E-02 ± 1.31E-04	1.36E-02 ± 1.17E-04	5.42E-03 ± 6.25E-05	4.21E-03 ± 7.69E-05	3.03E-03 ± 6.45E-05	8.71E-03 ± 2.61E-04
GJSD	1.45E-02 ± 1.14E-04	1.23E-02 ± 9.26E-05	4.86E-03 ± 4.52E-05	2.80E-03 ± 2.85E-05	2.57E-03 ± 3.50E-05	8.23E-03 ± 2.99E-04
IP-CAE	**1.36E-02** ± **5.35E-05**	**1.17E-02** ± **3.15E-05**	**4.38E-03** ± **1.28E-05**	**2.48E-03** ± **1.39E-05**	**2.03E-03** ± **2.75E-05**	**6.90E-03** ± **1.25E-04**

Table 2: **Classification Accuracy**. Top-1 accuracy for the classification task. The values are an average of 10 repetitions ± 1 standard deviation.

Model	MNIST	MNIST-Fashion	ISOLET	COIL-20	Smartphone HAR	Mice Protein
STG	92.29 ±0.30	80.85 ±0.27	84.95 ±0.31	96.80 ±0.25	88.80 ±0.08	68.24 ±1.11
LassoNet	90.06 ±0.33	78.28 ±0.36	84.33 ±0.28	89.37 ±0.47	92.44 ±0.11	77.12 ±0.80
CAE	83.10 ±1.23	73.19 ±0.82	75.82 ±2.31	80.70 ±2.93	82.72 ±0.80	63.10 ±6.51
GJSD	84.38 ±1.50	74.13 ±0.64	77.56 ±0.82	82.10 ±3.72	84.78 ±1.04	68.43 ±7.75
IP-CAE	**94.07** ±**0.37**	**82.68** ±**0.80**	**91.85** ±**0.55**	**97.92** ±**0.57**	**93.71** ±**0.62**	**94.26** ± **1.48**

ing channels in electroencephalogram (EEG) recordings. They employ a concrete layer for feature selection and propose a heuristic regularization to penalize duplicate channel selections and show its effectiveness. We found that its performance requires careful tuning of three hyperparameters.

Learning randomly perturbed structured predictors for direct loss minimization Indelman & Hazan (2021) propose learning the variance of the Gumbel noise perturbation in structured prediction. Although random perturbations are useful, they may mask the underlying signal during learning. By learning the variance of the perturbation noise, Indelman & Hazan (2021) achieve superior performance to both fixed and zero noise settings.

4. Experiments

In this section, we evaluate our proposed method on several datasets. Table 4 in Appendix A provides an overview of the datasets used. Across all experiments, we perform an ablation of the standard CAE, only IP, only the GJSD term, and both IP and the GJSD term. The hyperparameter P in $\Psi = \mathbb{R}^{K \times P}$ controlling the embedding dimension of the IP selector layer, is not tuned in our experiments. Instead, we simply set $P = D$, where D is the number of features for each dataset. The regularization strength hyperparameter for GJSD, λ, was tuned in $\{0, 0.0005, 0.005, 0.05\}$. Following CAE, we use a fixed learning rate of 0.001 with the Adam optimizer with moving-average coefficients $\boldsymbol{\beta} = (0.9, 0.999)$ and no weight decay, for all experiments and datasets. We train every model for 200 epochs, and select the weights corresponding to the best validation loss for test set evaluation. In all experiments, unless otherwise specified, we use an MLP with one hidden layer of 200 nodes for the decoder network. For the hidden activation, we use LeakyReLU with a slope of 0.2. For all experiments, we perform 10 repetitions and report the mean quantity. Any confidence intervals correspond to one standard deviation.

MNIST and MNIST-Fashion (LeCun et al., 1998; Xiao et al., 2017) consist of 28×28 greyscale images depicting digits and clothing items respectively. The supervised classification task is to predict the item from the pixel values.

ISOLET (Fanty & Cole, 1990) consists of preprocessed speech data of test subjects speaking all 52 letters of the English alphabet. The supervised classification task is to predict the spoken letter from the 617-dimensional speech data.

COIL-20 (Nene et al., 1996) consists of 32×32 greyscale images depicting 20 items, photographed on a rotating turntable at 5-degree increments (72 photos per item). The supervised classification task is to identify the item given the image.

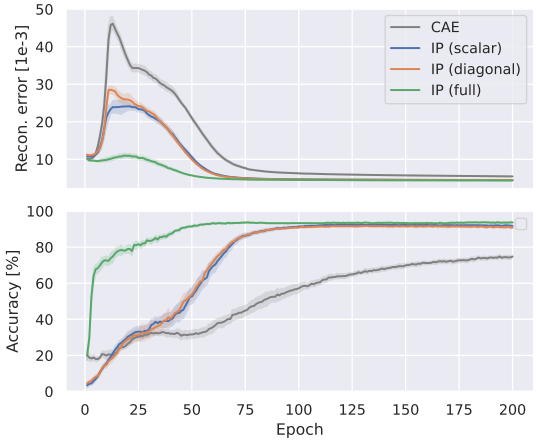


Figure 3: **IP parametrizations**. Validation results for CAE compared against three parametrizations of the linear IP weights on the ISOLET dataset.

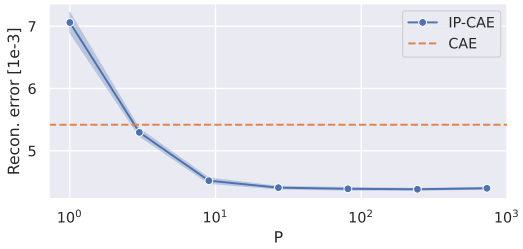


Figure 4: **Varying P**. Test set performance on ISOLET for varying size of IP P. The mean reconstruction error with CAE is included as a horizontal line.

Smartphone Dataset for Human Activity Recognition (Anguita et al., 2013) consists of sensor data collected from 30 subjects performing 6 activities while wearing a smartphone. The classification task is to predict the action from the sensor signals.

Mice Protein Expression (Higuera et al., 2015) consists of protein expressions from two groups of mice; control and trisomic mice. The supervised classification task is to predict the label consisting of the group, stimulation, and treatment of the mice. 7 proteins have missing values for one or more mice and these values were imputed with the average protein expression level of examples belonging to the same class of mice, following the dataset authors.

Reconstruction Error We train end-to-end and report the reconstruction error as the normalized Frobenius norm $\|X - \hat{X}\|_F / D$.

Classification Accuracy We train end-to-end and report the classification in a supervised setting and report the top-1 accuracy.

4.1. Improved Training and Generalization with IP

Figure 5 shows a significant improvement in training stability and convergence speed in both reconstruction and discriminative tasks. The curves represent our improved model (IP-CAE) with the original (CAE) across six common feature selection benchmarks. Each training was repeated for 10 random initializations, and the lines represent the mean validation loss (Figure 5a) and accuracy (Figure 5b), with one standard deviation indicated by the line width. Furthermore, IP-CAE converges to a lower validation error and higher accuracy than the original CAE. We also observe a significant speedup on all datasets (Table 3).

Table 3: **Speedup**. The mean speedup of IP-CAE compared to CAE, in terms of IP-CAE surpassing the performance of CAE (on validation data) trained for 200 epochs.

DATASET	RECON. ERROR	ACCURACY
MNIST	3.00×	4.00×
MNIST-FASHION	3.38×	4.60×
ISOLET	3.83×	18.77×
COIL-20	4.15×	25.68×
SMARTPHONE HAR	4.15×	3.70×
MICE PROTEIN	2.53×	12.92×

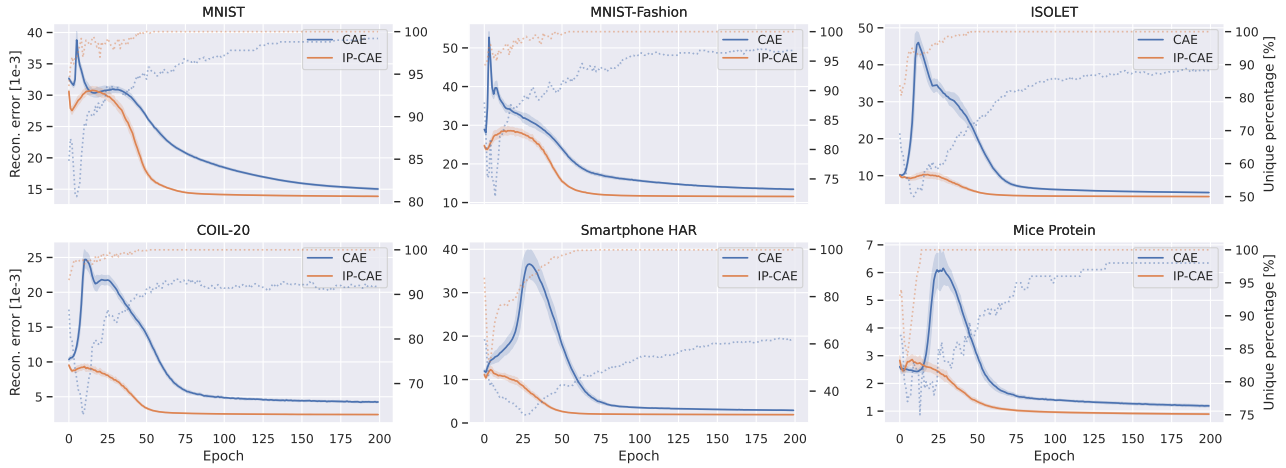
While we do not tune P, we include an ablation of the setting of P for one dataset, Isolet, in Figure 4. We conclude that the IP is not sensitive to a specific setting of P so long as it is sufficiently large, *i.e.* $P \approx D$. We find that the bias term in IP (Equation (6)) is redundant and does not affect performance.

To verify that this effect applies in general and not just for a specific setting of K, we vary K in $\{25, 50, 75, 100, 125\}$ on the ISOLET dataset. Figure 7 in Appendix B confirms that the results are valid regardless of K.

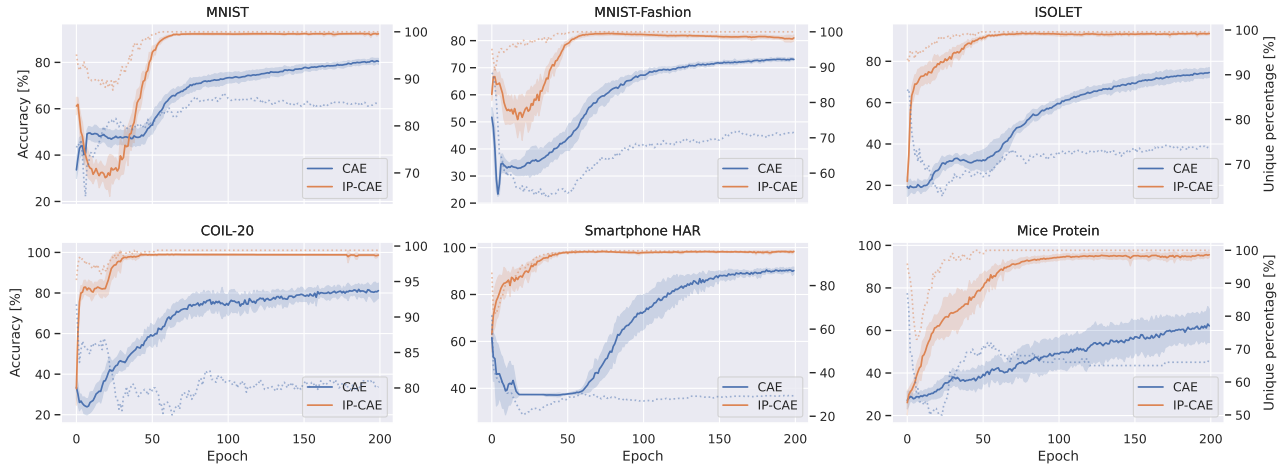
As a lower bound on performance, we include a comparison with the proposed GJSD regularization method which explicitly encourages unique selections. We find that such explicit encouragement outperforms CAE, but is not as effective as IP.

4.2. Special Cases of IP-CAE

As addressed in Section 2.4, CAE is a special case of IP-CAE with $\boldsymbol{W} = \mathbf{I}$ and $P = D$. Two additional special cases of IP-CAE that preserve the learning rate scaling properties (Appendix D) in the update rule while offering even simpler formulations are as follows:



(a) Reconstrucation error



(b) Accuracy

Figure 5: **Training Comparison**. Comparisons CAE and IP-CAE for (a) reconstruction error, (b) accuracy on the validation data throughout training. For IP-CAE, we let $P = D$. The mean unique percentages (definition 2.1) is shown by the dotted lines.

Single Scalar. The first alternative formulates $\boldsymbol{W}$ as $\boldsymbol{W} = w\boldsymbol{I}$, where w is a single learnable scalar parameter. This approach simplifies the complexity of $\boldsymbol{W}$ to a single degree of freedom.

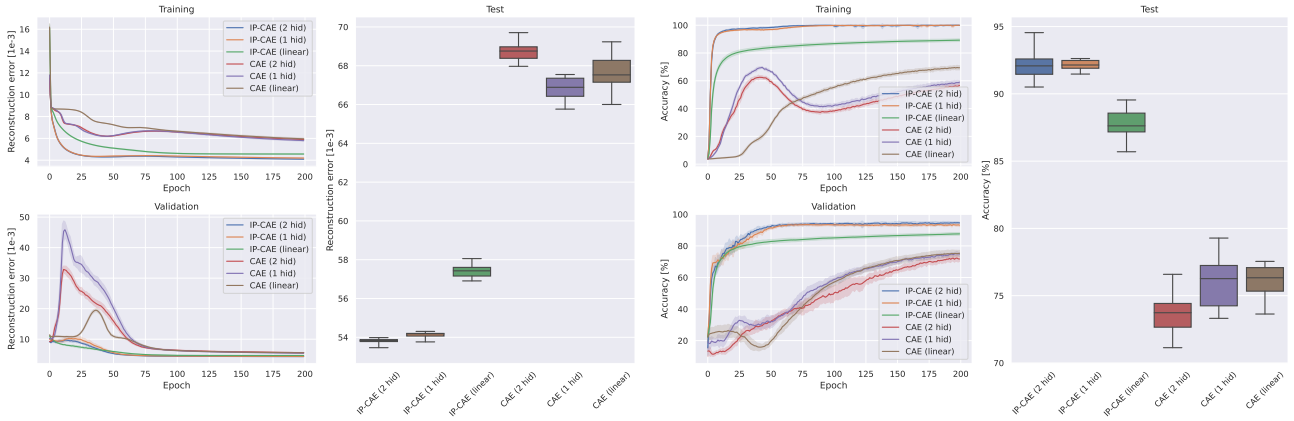
Diagonal Matrix. The second alternative represents $\boldsymbol{W}$ as $\boldsymbol{W} = \text{diag}(\mathbf{w})$, with $\mathbf{w}$ being a vector of learnable parameters.

Our empirical analysis, presented in Figure 3, contrasts these two simplified forms of $\boldsymbol{W}$ with the full matrix version and the standard CAE configuration. Both the scalar and diagonal versions demonstrate enhancements over CAE in stability and final performance. This improvement underscores the significance of the learning rate scaling property, as maintained in these simpler forms.

Most notably, allowing $\boldsymbol{W}$ to be a full matrix results in the most pronounced improvements in terms of training efficiency. This observation strongly suggests that the $\boldsymbol{W}\boldsymbol{W}^T$ term in the gradient transformation (Equation (9)) plays a significant role in the model's ability to learn complex embeddings to represent features.

4.3. Hidden Layers

It is worth noting that the main promise of neural network-based embedded feature selection is that it can be non-linear. We observe that for the original CAE, training is unstable with spikes in validation error, hindering a smooth convergence to the optimal solution. The problem worsens for decoders with multiple hidden layers. This is illustrated



(a) Reconstruction

(b) Classification

Figure 6: **Hidden Layers**. Results for varying decoder architectures on the ISOLET dataset, with and without IP. Three architectures are considered; linear, one hidden layer with 200 nodes, and two hidden layers with 200 nodes each. Unlike the original CAE, IP-CAE benefits significantly from additional decoder capacity. Boxes are quartiles and whiskers are min-max.

in Figure 6, where we investigate this issue for a linear decoder, an MLP decoder with one hidden layer, and an MLP decoder with two hidden layers. Our improved parameterization results in a smooth descent into a final validation loss that is lower than for the original CAE regardless of decoder complexity, and thus allows for the joint optimization of feature selection and non-linear decoders.

4.4. Comparison with STG and LassoNet

For completeness, we compare the test performance of the vanilla and IP-CAE with other baselines such as STG and LassoNet, in both reconstruction and classification settings in Tables 1 and 2. Unlike CAEs, both STG and LassoNet cannot optimize for a specific number of selected features directly, instead, they require the hyperparameter λ, which denotes the strength of regularization, to be specified, which in turn affects the number of selected features. LassoNet also requires an additional hyperparameter M, which denotes the hierarchy coefficient, however, following (Lemhadri et al., 2021) we use $M = 10$ for all the datasets. We run extensive ablations over the hyperparameter λ and choose the value that returns 50 feature selections for all our datasets (with the exception of MICE, where we have 10 feature selections). For STG and LassoNet, we use a single hidden layer MLP with 200 dimensions and ReLU activation during feature selection. Later, we retrain a single hidden layer MLP with 200 dimensions and ReLU from scratch to report the test accuracy, for each dataset.

We used the official code repositories of STG and LassoNet for all our experiments. For STG, during and after feature selection, the networks were trained for 100 epochs for all datasets. For LassoNet, during feature selection, we used the default setting from the official repository to train the initial dense network for 1000 epochs followed by 100 epochs of training for each sparse network in the iterative process (increasing λ). After feature selection, we again run 100 epochs of network training with the corresponding selected features for all datasets.

For both CAE and IP-CAE, we refrain from retraining the decoder, but instead directly evaluate it using the jointly learned decoder. We chose this approach because it aligns with the core principle of embedded feature selection, which is to utilize features non-linearly and jointly optimize for feature selection and the non-linear training objective.

We demonstrate that CAE falls behind STG and LassoNet by a significant margin both for reconstruction and classification. But IP-CAE significantly outperforms them in test accuracy (Table 2) and reconstruction error (Table 1) on all datasets. Additionally, being a stochastic method, CAE is prone to high variance. This problem is reduced slightly in IP-CAE on most datasets, as shown in Figure 5.

5. Discussion

In this paper, we addressed the practical challenges of training CAEs. We proposed IP-CAEs which implicitly alleviate redundant features and instability. Our approach achieves state-of-the-art reconstruction error and accuracy for all datasets considered, up to 20 times faster than vanilla CAE.

While, in this paper, we establish the empirical effectiveness of IP-CAE across a wide range of datasets and tasks, and argue for it from the lens of implicit overparametrization,

the remarkable results motivates the need for a more formal study as a future direction.

With the goal of understanding IP-CAE's success, we introduced GJSD regularization, which forces unique selections. This baseline significantly improves CAE in every dataset and task, but falls behind IP, as well as LassoNet and STG. This, interestingly, indicates that the effect of IP cannot be solely attributed to removing duplicate features. IP and GJSD are not exclusive and can be combined, but we found that adding GJSD regularization to IP-CAE not to improve results significantly.

Finally, the IP method we present is, in principle, generalizable to Gumbel-Softmax distributions beyond feature selection which is left for future work.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

Acknowledgements

We thank the anonymous reviewers for their valuable feedback and suggestions. This work was partially supported by KTH Digital Futures, the Swedish eScience Research Centre (SeRC), the Swedish Foundation for Strategic Research grants BD15-0043 and ID19-0052, the Marie Skłodowska-Curie Actions project "MODELAIR" through grant no. 101072559, and Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The computations were enabled by Alvis cluster access provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) at Chalmers University of Technology partially funded by the Swedish Research Council through grant agreement no. 2022-06725 as well as the Berzelius resource provided by the Knut and Alice Wallenberg Foundation at the National Supercomputer Centre.

References

Amaldi, E. and Kann, V. On the Approximability of Minimizing Nonzero Variables or Unsatisfied Relations in Linear Systems. *Theoretical Computer Science*, 209: 237–260, 1998.

Anguita, D., Ghio, A., Oneto, L., Parra, X., and Reyes-Ortiz, J. L. A Public Domain Dataset for Human Activity Recognition using Smartphones. In *The European Symposium on Artificial Neural Networks*, 2013.

Bakker, T., van Hoof, H., and Welling, M. Experimental Design for MRI by Greedy Policy Search. In *Advances in Neural Information Processing Systems*, 2020.

Balın, M. F., Abid, A., and Zou, J. Concrete Autoencoders: Differentiable Feature Selection and Reconstruction. In *International Conference on Machine Learning*, 2019.

Barber, R. F. and Candès, E. J. Controlling the False Discovery Rate via Knockoffs. *The Annals of Statistics*, 43, 2015.

Cai, J., Luo, J., Wang, S., and Yang, S. Feature Selection in Machine Learning: A New Perspective. *Neurocomputing*, 300:70–79, 2018.

Carvalho, C. M., Polson, N. G., and Scott, J. G. Handling Sparsity via the Horseshoe. In *International Conference on Artificial Intelligence and Statistics*, 2009.

Chen, J., Song, L., Wainwright, M., and Jordan, M. Learning to Explain: An Information-Theoretic Perspective on Model Interpretation. In *International Conference on Machine Learning*, 2018.

Cherepanova, V., Levin, R., Somepalli, G., Geiping, J., Bruss, C. B., Wilson, A. G., Goldstein, T., and Goldblum, M. A Performance-Driven Benchmark for Feature Selection in Tabular Deep Learning. In *Advances in Neural Information Processing Systems*, 2023.

Cui, C. and Wang, D. High Dimensional Data Regression Using Lasso Model and Neural Networks with Random Weights. *Information Sciences*, 372:505–517, 2016.

Englesson, E. and Azizpour, H. Generalized Jensen-Shannon Divergence Loss for Learning with Noisy Labels. *Advances in Neural Information Processing Systems*, 2021.

Fanty, M. and Cole, R. Spoken Letter Recognition. In *Advances in Neural Information Processing Systems*, 1990.

Gumbel, E. J. The Maxima of the Mean Largest Value and of the Range. *The Annals of Mathematical Statistics*, 25: 76–84, 1954.

Guyon, I. and Elisseeff, A. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

Hendrycks, D., Mu, N., Cubuk, E. D., Zoph, B., Gilmer, J., and Lakshminarayanan, B. AugMix: A Simple Data Processing Method to Improve Robustness and Uncertainty. In *International Conference on Learning Representations*, 2019.

Higuera, C., Gardiner, K. J., and Cios, K. J. Self-Organizing Feature Maps Identify Proteins Critical to Learning in a Mouse Model of Down Syndrome. *PLoS ONE*, 10, 2015.

Huijben, I. A. M., Veeling, B. S., and van Sloun, R. J. G. Deep probabilistic subsampling for task-adaptive compressed sensing. In *International Conference on Learning Representations*, 2019.

Indelman, H. C. and Hazan, T. Learning Randomly Perturbed Structured Predictors for Direct Loss Minimization. In *International Conference on Machine Learning*, 2021.

Jang, E., Gu, S., and Poole, B. Categorical Reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations*, 2017.

Kviman, O., Melin, H., Koptagel, H., Elvira, V., and Lagergren, J. Multiple Importance Sampling ELBO and Deep Ensembles of Variational Approximations. In *International Conference on Artificial Intelligence and Statistics*, 2022.

Kviman, O., Molén, R., Hotti, A., Kurt, S., Elvira, V., and Lagergren, J. Cooperation in the Latent Space: The Benefits of Adding Mixture Components in Variational Autoencoders. In *International Conference on Machine Learning*, 2023.

LeCun, Y., Cortes, C., and Burges, C. The MNIST Database of Handwritten Digits. 1998.

Lemhadri, I., Ruan, F., and Tibshirani, R. LassoNet: Neural Networks with Feature Sparsity. In *International Conference on Artificial Intelligence and Statistics*, 2021.

Li, J., Cheng, K., Wang, S., Morstatter, F., Trevino, R. P., Tang, J., and Liu, H. Feature Selection: A Data Perspective. *ACM Computing Surveys*, 50:94:1–94:45, 2017.

Louizos, C., Welling, M., and Kingma, D. P. Learning Sparse Neural Networks through L_0 Regularization. In *International Conference on Learning Representations*, 2018.

Maddison, C. J., Mnih, A., and Teh, Y. W. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *International Conference on Learning Representations*, 2017.

Nene, S. A., Nayar, S. K., Murase, H., et al. Columbia Object Image Library (COIL-20). *Technical Report CUCS-005-96*, 1996.

Ročková, V. and George, E. I. The Spike-and-Slab LASSO. *Journal of the American Statistical Association*, 113:431–444, 2018.

Romano, Y., Sesia, M., and Candès, E. Deep Knockoffs. *Journal of the American Statistical Association*, 115:1861–1872, 2020.

Sokar, G., Atashgahi, Z., Pechenizkiy, M., and Mocanu, D. C. Where to Pay Attention in Sparse Training for Feature Selection? *Advances in Neural Information Processing Systems*, 2022.

Strypsteen, T. and Bertrand, A. End-to-End Learnable EEG Channel Selection for Deep Neural Networks with Gumbel-Softmax. *Journal of Neural Engineering*, 18, 2021.

Tibshirani, R. Regression Shrinkage and Selection Via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58:267–288, 1996.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Yamada, Y., Lindenbaum, O., Negahban, S., and Kluger, Y. Feature Selection using Stochastic Gates. In *International Conference on Machine Learning*, 2020.

Yoon, J., Jordon, J., and van der Schaar, M. INVASE: Instance-wise Variable Selection using Neural Networks. In *International Conference on Learning Representations*, 2018.

A. Experimental Details

In this appendix, we provide a detailed description of the experimental setup.

A.1. Datasets

Table 4 provides an overview of all datasets used in experiments, the type of data, number of samples N, features D, selected features K, and classes C.

A.2. Clarification of hyperparameters

We use the same temperature annealing schedule as CAE, and the same maximum temperature $T_0 = 10$ and minimum temperature $T_B = 0.01$. We searched the regularization strength hyperparameter of the GJSD term in {5.00E-04, 5.00E-03, 5.00E-02} for both reconstruction and classification. The optimal settings found are listed in Table 5.

A.3. Code

The source code is available at `https://github.com/Alfred-N/IP-CAE`.

We have taken considerable care to ensure the ease of reproducibility of all our results. For each dataset, we have included a configuration file named `<dataset>/base.yaml` which contains the necessary hyperparameters to run CAE exactly as we did in our paper for the reconstruction task. To run with our proposed IP, simply specify the `--dim_ip` optional argument when executing our training script, which refers to the dimensionality of the IP vectors, namely P. Example: `python src/main_pl.py --config=configs/ISOLET/base.yaml --dim_ip=617`. The setting of `dim_ip` (P) used throughout all our experiments with IP was configured to match the feature dimensions of each dataset, D, which can be found in Table 1 of the main report. Similarly, the corresponding configs for the classification task can be found as `<dataset>/classification.yaml`.

We log training metrics with WandB. To start tracking without a WandB account, simply run `src/main_pl.py` and select option `(1)Private W&B dashboard, no account required` when prompted. Note that this requires an internet connection. To run offline, select option `(4)Don't visualize my results`.

To further facilitate reproducibility, we include a script, `src/fs_datasets.py`, for downloading all datasets used in this paper, which includes functions that return the exact train/test/validation splits that were used. The data will be automatically downloaded into `--data_root_dir` when running `src/main_pl.py`.

For all of our experiments that we repeated for 10 seeds, we used fixed seeds {11, 22, 33, 44, 55, 66, 77, 88, 99, 1010}. Thus, all results can be reproduced exactly and deterministically if specifying the (integer) argument `--seed`.

Finally, `--IP_weights` flag can be used to specify different weight options of IP such as `scalar`, `diag` or `shared`. Note that the general version of IP we describe in the main paper refers to the `shared` option.

A.4. Data and preprocessing

For COIL-20 we use the version of the dataset provided by (Li et al., 2017). For MNIST and Fashion-MNIST, we use the versions provided in Torchvision. For the other datasets, ISOLET, Smartphone HAR, and Mice protein we use the version provided at UCI (Fanty & Cole, 1990; Anguita et al., 2013; Higuera et al., 2015).

We identified a potential bug in the preprocessing of the Mice Protein dataset used by CAE. They impute missing values with a "filling value" of -10^5 and then take column averages of each protein expression and replace the filling value. The expression levels are generally in the order of magnitude of 10^0 to 10^1, which means the average is dominated by the filling value rather than the signal. Additionally, they overwrite the same array they use to calculate averages on the fly instead of inputting the data in a new array.

We instead use the imputation method described by the authors of the Mice Protein dataset (Higuera et al., 2015), which means averaging missing protein expression values with the average expression corresponding to that protein for the same class of mice.

Additionally, CAE computes their min-max scaling based on the statistics of the full dataset. We instead calculate the min-max scaling statistics only on the training split and then use them to scale the validation and test split accordingly.

A.5. Compute infrastructure

We used an external cluster with T4 and A40 GPUs. Each model was trained on a single GPU.

B. Additional Experiments

In this section, we provide additional experiments that were left out of the paper due to the space limit.

B.1. Extended training

We compare the convergence with an increased number of epochs in the ISOLET dataset. We increase the number of epochs to 1000 (from 200 in our other experiments). This way, the annealing schedule of the temperature is stretched over a longer period, which means a longer exploration phase with high randomness.

Table 4: **Datasets**. An overview of the datasets used. The number of samples N includes both training and test data. For MNIST and MNIST-Fashion, the data used is a random subset of the full data.

NAME	INPUT TYPE	SAMPLES (N)	FEATURES (D)	SELECTED (K)	CLASSES (C)
MNIST	GRAYSCALE IMAGE	10500	784	50	10
MNIST-FASHION	GRAYSCALE IMAGE	10500	784	50	10
ISOLET	SPEECH	7797	617	50	26
COIL-20	GRAYSCALE IMAGE	1440	1024	50	20
SMARTPHONE HAR	SENSOR TIME SERIES	10299	561	50	6
MICE PROTEIN	PROTEIN EXPRESSION	1080	77	10	8

Table 5: **GJSD settings**. Optimal settings of the GJSD regularization strength hyperparameter λ using the original CAE parametrization.

DATASET	CLASSIFICATION	RECONSTRUCTION
MNIST	5.00E-02	5.00E-02
MNIST-FASHION	5.00E-02	5.00E-02
SMARTPHONE HAR	5.00E-02	5.00E-03
COIL-20	5.00E-02	5.00E-02
ISOLET	5.00E-02	5.00E-03
MICE PROTEIN	5.00E-02	5.00E-03

Table 6: **Extended Training**. The mean test set performance on ISOLET for CAE and IP-CAE trained for 200 and 1000 epochs. The mean is computed using ten repetitions.

MODEL	EPOCHS	RECON. ERROR	ACCURACY
CAE	200	0.067	75.8
CAE	1000	0.054	91.0
IP-CAE	200	0.054	**91.9**
IP-CAE	1000	**0.053**	91.4

As mentioned by the CAE authors: *"if the temperature is held low, the concrete selector layer is not able to explore different combinations of features and converges to a poor local minimum"*.

We find that this longer training drastically improves CAE, which converges to higher accuracy, lower reconstruction error and higher unique percentage, see Table 6. However, IP-CAE trained for 200 epochs still outperforms CAE trained for 1000 epochs. We emphasize that this is for illustrative purposes. Training for five times as many epochs is not an efficient solution to CAEs' undesirable training behavior. Interestingly, we observe that CAE does not achieve 100% unique selections for classification on the ISOLET dataset, which seems to limit the resulting accuracy.

B.2. Number of selected features

B.3. Embedding dimensionality

Here, we provide additional experiments showcasing the effect of the parameter P on the ISOLET dataset. As evident from the results, IP-CAE outperforms CAE even when using a smaller number of parameters. We call this setting *underparameterized*. Naturally, as the number of parameters decreases further, the IP-CAE performs worse than the CAE at a certain point. The training seems to improve with P, but not necessarily the end result if trained to convergence.

Table 7: **Runtime**. The average time in minutes of training for 200 epochs, with and without IP.

DATASET	CLASSIFICATION		RECONSTR.	
	CAE	IP-CAE	CAE	IP-CAE
MNIST	8.55	8.73	8.74	8.86
MNIST-FASHION	8.67	8.82	8.71	8.87
SMART. HAR	5.47	5.71	5.07	5.15
COIL-20	5.28	5.67	4.95	5.40
ISOLET	6.10	6.21	5.85	5.94
MICE PROTEIN	4.14	4.41	3.55	3.63

B.4. Learning rate warmup

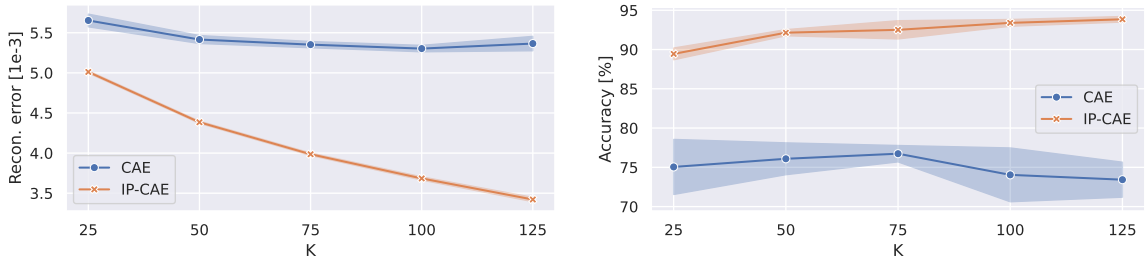
We use a warmup phase with a linearly increasing learning rate from 10^{-6} to 10^{-3} for the first $\{25, 50, 75, 100\}$ epochs out of 200. This reduces the spike in validation loss but leads to worse results (Figure 9a). Our interpretation is that the lower learning rate causes the model to learn less during the critical early exploration phase, a phase that is critical to finding optimal minima (Balın et al., 2019).

B.5. Weight decay

We consider weight decay with parameters $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$, all of which yield worse results (Figure 9b). This is consistent with Bora et al. (2019), where no weight decay is used.

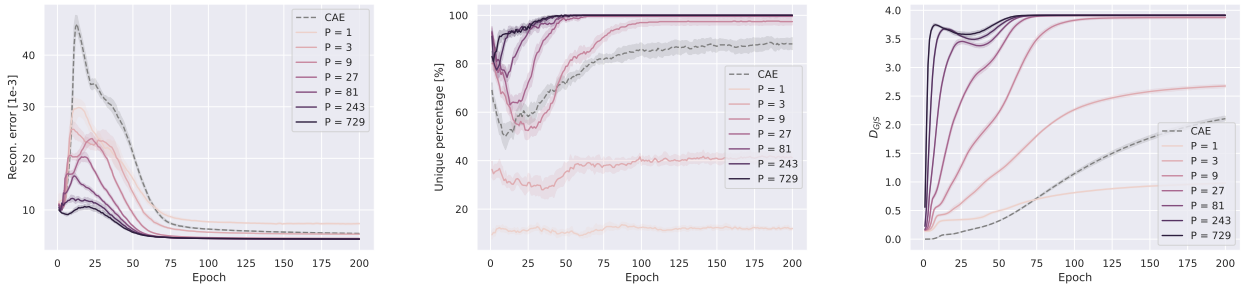
B.6. Runtime

We report the average run time on a T40 GPU in Table 7.



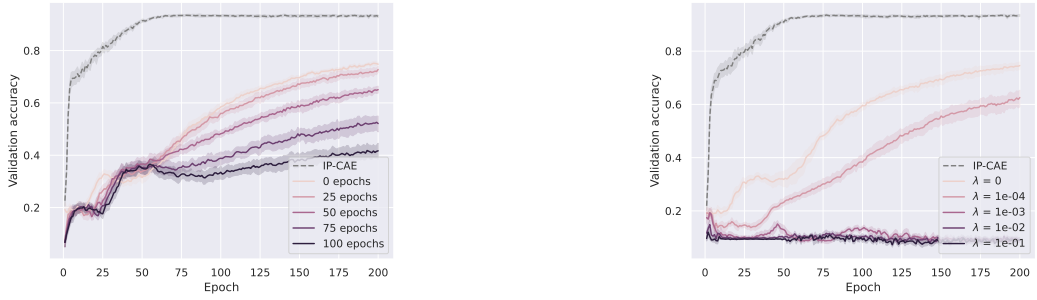
(a) Reconstruction (b) Classification

Figure 7: **Varying K**. Test set performance for (a) reconstruction and (b) classification while varying the number of features selected K with and without IP on ISOLET.



(a) Reconstruction error (b) Unique percentage (c) D_{GJS}

Figure 8: **Convergence with varying P**. Validation results for a) reconstruction error, b) unique percentage, and c) generalized Jensen-Shannon divergence. Plots are the mean of ten repetitions and confidence bounds show one standard deviation.

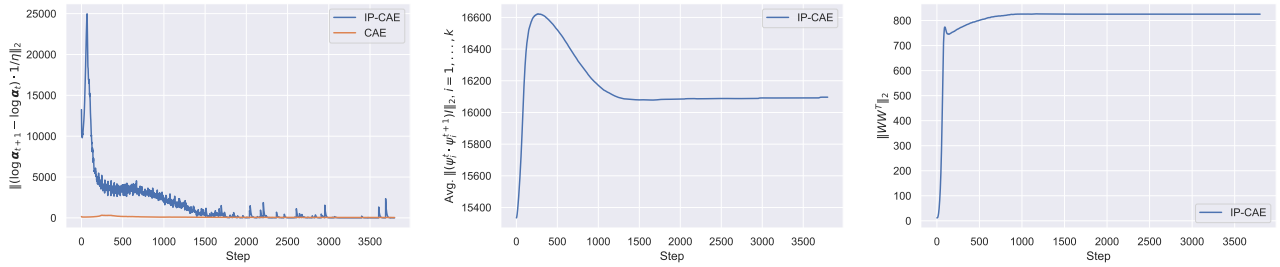


(a) Learning rate warmup (b) Weight decay

Figure 9: **Conventional training modifications**. An interesting question is whether the problems discussed in this work can be alleviated using conventional techniques like scheduling the learning rate (a) or through standard weight decay regularization (b).

C. IP-CAE Updates

In this appendix, we include additional plots showing how components of the IP-CAE update change over time (Figure 10).



(a) $\boldsymbol{\alpha}$ (b) ψ (c) $\boldsymbol{W}$

Figure 10: **IP-CAE Update**. Components of the update rule related to (a) $\boldsymbol{\alpha}$, (b) ψ, and (c) $\boldsymbol{W}$ for each step throughout training for classification on ISOLET. In (a), the value is compared to its equivalent in the vanilla CAE.

D. Update Rules for IP-CAE

In this appendix, we derive the update rules for IP-CAE.

Full Weight Matrix. With a full $\boldsymbol{W}$ matrix, we have $\log \boldsymbol{\alpha}_i^{(t+1)} = \boldsymbol{W}^{(t+1)} \boldsymbol{\psi}_i^{(t+1)}$ and:

$$\boldsymbol{W}^{(t+1)} \leftarrow \boldsymbol{W} - \eta \nabla_{\boldsymbol{W}} \mathcal{L} \tag{12}$$

$$\boldsymbol{\psi}_i^{(t+1)} \leftarrow \boldsymbol{\psi}_i - \eta \nabla_{\boldsymbol{\psi}_i} \mathcal{L} \tag{13}$$

Thus, to derive the update rule for $\log \boldsymbol{\alpha}_i^{(t+1)}$ we need to first derive $\nabla_{\boldsymbol{W}} \mathcal{L}$ and $\nabla_{\boldsymbol{\psi}_i} \mathcal{L}$, which are the gradients of the loss with respect to $\boldsymbol{W}$ and $\boldsymbol{\psi}_i$, respectively. We have:

$$\nabla_{\boldsymbol{W}} \mathcal{L} = \begin{bmatrix} \nabla_{\boldsymbol{W}_{1,1}} \mathcal{L} & \dots & \nabla_{\boldsymbol{W}_{1,D}} \mathcal{L} \\ \vdots & & \\ \nabla_{\boldsymbol{W}_{D,1}} \mathcal{L} & \dots & \nabla_{\boldsymbol{W}_{D,D}} \mathcal{L} \end{bmatrix} \tag{14}$$

$$= \begin{bmatrix} (\nabla_{\boldsymbol{W}\boldsymbol{\psi}_i} \mathcal{L})^T (\nabla_{\boldsymbol{W}_{1,1}} \boldsymbol{W}\boldsymbol{\psi}_i)^T & \dots & (\nabla_{\boldsymbol{W}\boldsymbol{\psi}_i} \mathcal{L})^T (\nabla_{\boldsymbol{W}_{1,D}} \boldsymbol{W}\boldsymbol{\psi}_i)^T \\ \vdots & & \\ (\nabla_{\boldsymbol{W}\boldsymbol{\psi}_i} \mathcal{L})^T (\nabla_{\boldsymbol{W}_{D,1}} \boldsymbol{W}\boldsymbol{\psi}_i)^T & \dots & (\nabla_{\boldsymbol{W}\boldsymbol{\psi}_i} \mathcal{L})^T (\nabla_{\boldsymbol{W}_{D,D}} \boldsymbol{W}\boldsymbol{\psi}_i)^T \end{bmatrix} \tag{15}$$

$$= \begin{bmatrix} (\nabla_{\boldsymbol{W}\boldsymbol{\psi}_i} \mathcal{L})_1^T \boldsymbol{\psi}_{i,1} & \dots & (\nabla_{\boldsymbol{W}\boldsymbol{\psi}_i} \mathcal{L})_1^T \boldsymbol{\psi}_{i,D} \\ \vdots & & \\ (\nabla_{\boldsymbol{W}\boldsymbol{\psi}_i} \mathcal{L})_D^T \boldsymbol{\psi}_{i,1} & \dots & (\nabla_{\boldsymbol{W}\boldsymbol{\psi}_i} \mathcal{L})_D^T \boldsymbol{\psi}_{i,D} \end{bmatrix} \tag{16}$$

$$= (\nabla_{\boldsymbol{W}\boldsymbol{\psi}_i} \mathcal{L}) \boldsymbol{\psi}_i^T \tag{17}$$

The step between Equation (15) and Equation (16) becomes clearer by studying the gradient $\nabla_{\boldsymbol{W}_{j,k}} \boldsymbol{W}\boldsymbol{\psi}_i$:

$$(\nabla_{\boldsymbol{W}_{j,k}} \boldsymbol{W}\boldsymbol{\psi}_i)_l = \nabla_{\boldsymbol{W}_{j,k}} \sum_{m=1}^{D} \boldsymbol{W}_{l,m} \boldsymbol{\psi}_{i,m} = \begin{cases} \boldsymbol{\psi}_{i,k}, & \text{if } j \text{ and } l \text{ are the same} \\ 0, & \text{otherwise} \end{cases} \tag{18}$$

Thus, the full gradient $\nabla_{\boldsymbol{W}_{j,k}} \boldsymbol{W}\boldsymbol{\psi}_i \in \mathbb{R}^D$ is a vector with all zeros except in component j where it is $\boldsymbol{\psi}_{i,k}$. Thus, the dot product $(\nabla_{\boldsymbol{W}\boldsymbol{\psi}_i} \mathcal{L})^T (\nabla_{\boldsymbol{W}_{j,k}} \boldsymbol{W}\boldsymbol{\psi}_i)^T$ will only have a single non-zero term corresponding to $(\nabla_{\boldsymbol{W}\boldsymbol{\psi}_i} \mathcal{L})_j \boldsymbol{\psi}_{i,k}$.

Similarly, for $\nabla_{\boldsymbol{\psi}_i}\mathcal{L}$, we have:

$$\nabla_{\boldsymbol{\psi}_i}\mathcal{L} = \begin{bmatrix} \nabla_{\boldsymbol{\psi}_{i,1}}\mathcal{L} \\ \vdots \\ \nabla_{\boldsymbol{\psi}_{i,D}}\mathcal{L} \end{bmatrix} \tag{19}$$

$$= \begin{bmatrix} (\nabla_{\boldsymbol{W}\boldsymbol{\psi}_i}\mathcal{L})^T(\nabla_{\boldsymbol{\psi}_{i,1}}\boldsymbol{W}\boldsymbol{\psi}_i)^T \\ \vdots \\ (\nabla_{\boldsymbol{W}\boldsymbol{\psi}_i}\mathcal{L})^T(\nabla_{\boldsymbol{\psi}_{i,D}}\boldsymbol{W}\boldsymbol{\psi}_i)^T \end{bmatrix} \tag{20}$$

$$= \begin{bmatrix} (\nabla_{\boldsymbol{W}\boldsymbol{\psi}_i}\mathcal{L})^T[\boldsymbol{W}_{1,1},\ldots,\boldsymbol{W}_{D,1}]^T \\ \vdots \\ (\nabla_{\boldsymbol{W}\boldsymbol{\psi}_i}\mathcal{L})^T[\boldsymbol{W}_{1,D},\ldots,\boldsymbol{W}_{D,D}]^T \end{bmatrix} \tag{21}$$

$$= \boldsymbol{W}^T(\nabla_{\boldsymbol{W}\boldsymbol{\psi}_i}\mathcal{L}) \tag{22}$$

Now, we can express the update rule:

$$\log \boldsymbol{\alpha}_i^{(t+1)} = \boldsymbol{W}^{(t+1)}\boldsymbol{\psi}_i^{(t+1)} \tag{23}$$

$$= (\boldsymbol{W} - \eta\nabla_{\boldsymbol{W}}\mathcal{L})(\boldsymbol{\psi}_i - \eta\nabla_{\boldsymbol{\psi}_i}\mathcal{L}) \tag{24}$$

$$= (\boldsymbol{W} - \eta(\nabla_{\boldsymbol{W}\boldsymbol{\psi}_i}\mathcal{L})\boldsymbol{\psi}_i^T)(\boldsymbol{\psi}_i - \eta\boldsymbol{W}^T(\nabla_{\boldsymbol{W}\boldsymbol{\psi}_i}\mathcal{L})) \tag{25}$$

$$= \boldsymbol{W}\boldsymbol{\psi}_i - \eta\boldsymbol{W}\boldsymbol{W}^T(\nabla_{\boldsymbol{W}\boldsymbol{\psi}_i}\mathcal{L}) \tag{26}$$

$$- \eta(\nabla_{\boldsymbol{W}\boldsymbol{\psi}_i}\mathcal{L})\boldsymbol{\psi}_i^T\boldsymbol{\psi}_i + \eta^2(\nabla_{\boldsymbol{W}\boldsymbol{\psi}_i}\mathcal{L})\boldsymbol{\psi}_i^T\boldsymbol{W}^T(\nabla_{\boldsymbol{W}\boldsymbol{\psi}_i}\mathcal{L}) \tag{27}$$

$$= \boldsymbol{W}\boldsymbol{\psi}_i - \eta(\boldsymbol{W}\boldsymbol{W}^T + \boldsymbol{\psi}_i^T(\boldsymbol{\psi}_i - \eta\boldsymbol{W}^T(\nabla_{\boldsymbol{W}\boldsymbol{\psi}_i}\mathcal{L}))\boldsymbol{I})(\nabla_{\boldsymbol{W}\boldsymbol{\psi}_i}\mathcal{L}) \tag{28}$$

$$= \boldsymbol{W}\boldsymbol{\psi}_i - \eta\boldsymbol{T}_i\nabla_{\boldsymbol{W}\boldsymbol{\psi}_i}\mathcal{L} \tag{29}$$

which is the same as in Equation 9.

Scalar Weight. With a scalar weight w, we have $\log \boldsymbol{\alpha}_i^{(t+1)} = w^{(t+1)}\boldsymbol{\psi}_i^{(t+1)}$ and:

$$w^{(t+1)} \leftarrow w - \eta\nabla_w\mathcal{L} \tag{30}$$

$$\boldsymbol{\psi}_i^{(t+1)} \leftarrow \boldsymbol{\psi}_i - \eta\nabla_{\boldsymbol{\psi}_i}\mathcal{L} \tag{31}$$

Thus, to derive the update rule for $\log \boldsymbol{\alpha}_i^{(t+1)}$ we need to first derive $\nabla_w\mathcal{L}$ and $\nabla_{\boldsymbol{\psi}_i}\mathcal{L}$. We have:

$$\nabla_w\mathcal{L} = (\nabla_{w\boldsymbol{\psi}_i}\mathcal{L})^T(\nabla_w w\boldsymbol{\psi}_i) = (\nabla_{w\boldsymbol{\psi}_i}\mathcal{L})^T\boldsymbol{\psi}_i \tag{32}$$

and:

$$\nabla_{\boldsymbol{\psi}_i}\mathcal{L} = \begin{bmatrix} \nabla_{\boldsymbol{\psi}_{i,1}}\mathcal{L} \\ \vdots \\ \nabla_{\boldsymbol{\psi}_{i,D}}\mathcal{L} \end{bmatrix} \tag{33}$$

$$= \begin{bmatrix} (\nabla_{w\boldsymbol{\psi}_i}\mathcal{L})^T(\nabla_{\boldsymbol{\psi}_{i,1}}w\boldsymbol{\psi}_i)^T \\ \vdots \\ (\nabla_{w\boldsymbol{\psi}_i}\mathcal{L})^T(\nabla_{\boldsymbol{\psi}_{i,D}}w\boldsymbol{\psi}_i)^T \end{bmatrix} \tag{34}$$

$$= \begin{bmatrix} (\nabla_{w\boldsymbol{\psi}_i}\mathcal{L})^T[w,0,\ldots,0]^T \\ \vdots \\ (\nabla_{w\boldsymbol{\psi}_i}\mathcal{L})^T[0,\ldots,0,w]^T \end{bmatrix} \tag{35}$$

$$= w(\nabla_{w\boldsymbol{\psi}_i}\mathcal{L}) \tag{36}$$

Now, we can express the update rule:

$$\begin{aligned}
\log \boldsymbol{\alpha}_i^{(t+1)} &= w^{(t+1)} \boldsymbol{\psi}_i^{(t+1)} && (37)\\
&= (w - \eta \nabla_w \mathcal{L})(\boldsymbol{\psi}_i - \eta \nabla_{\boldsymbol{\psi}_i} \mathcal{L}) && (38)\\
&= (w - \eta (\nabla_{w\boldsymbol{\psi}_i} \mathcal{L})^T \boldsymbol{\psi}_i)(\boldsymbol{\psi}_i - \eta w (\nabla_{w\boldsymbol{\psi}_i} \mathcal{L})) && (39)\\
&= w\boldsymbol{\psi}_i - \eta w^2 (\nabla_{w\boldsymbol{\psi}_i} \mathcal{L}) && (40)\\
&\quad - \eta ((\nabla_{w\boldsymbol{\psi}_i} \mathcal{L})^T \boldsymbol{\psi}_i)\boldsymbol{\psi}_i + \eta^2 w ((\nabla_{w\boldsymbol{\psi}_i} \mathcal{L})^T \boldsymbol{\psi}_i)(\nabla_{w\boldsymbol{\psi}_i} \mathcal{L}) && (41)\\
&= w\boldsymbol{\psi}_i - \eta (w^2 - \eta w (\nabla_{w\boldsymbol{\psi}_i} \mathcal{L})^T \boldsymbol{\psi}_i)(\nabla_{w\boldsymbol{\psi}_i} \mathcal{L}) - \eta ((\nabla_{w\boldsymbol{\psi}_i} \mathcal{L})^T \boldsymbol{\psi}_i)\boldsymbol{\psi}_i && (42)\\
&= w\boldsymbol{\psi}_i - \eta w (w - \eta \nabla_w \mathcal{L})(\nabla_{w\boldsymbol{\psi}_i} \mathcal{L}) - \eta (\nabla_w \mathcal{L})\boldsymbol{\psi}_i && (43)\\
&= w\boldsymbol{\psi}_i - \eta (w w^{(t+1)} (\nabla_{w\boldsymbol{\psi}_i} \mathcal{L}) + (\nabla_w \mathcal{L})\boldsymbol{\psi}_i) && (44)
\end{aligned}$$

Thus, with a scalar weight w the update rule takes on a slightly different form compared to the full matrix $\boldsymbol{W}$. One interpretation is that the standard gradients $\nabla_{w\boldsymbol{\psi}_i} \mathcal{L}$ are still scaled, but now by $w^{(t)} w^{(t+1)}$ and furthermore, differently from the full matrix case, the gradients are also translated by $(\nabla_w \mathcal{L})\boldsymbol{\psi}_i$.