Chapter 5

Deep Representations as Unrolled Optimization

"*What I cannot create, I do not understand.*"

— Richard Feynman

In previous chapters, we have shown how to identify low-dimensional structures in high-dimensional spaces, *mainly focusing on linear structures*. For example, we introduced principal component analysis (PCA) to learn the linear denoiser $\hat{\boldsymbol{U}}^\top$ when the observed data $\boldsymbol{x}$ follow the statistical model $\boldsymbol{x} = \boldsymbol{U}\boldsymbol{z} + \boldsymbol{\varepsilon}$. In this setting, the learned representations are linearly transformed input data $\hat{\boldsymbol{U}}^\top \boldsymbol{x}$. Under the linear model assumption, one can learn the low-dimensional linear structure with efficient optimization algorithms and strong theoretical guarantees. Moreover, the linear model assumption covers a wide range of applications and problems, including face recognition, magnetic resonance image recovery, and structure texture recovery [WM22].

On the other hand, the linear model can be limited when dealing with real-world applications, especially when the input data $\boldsymbol{x}$ is complex, such as speech and natural languages, images and videos, and robotic motions. The low-dim distributions of such data are typically *nonlinear*. How to deal with nonlinearity has a long history across different disciplines such as control theory, signal processing, and pattern recognition. There have been considerable efforts that try to extend methods and solutions for linear models to handle nonlinearity, including early effort to extend PCA to nonlinear PCA (as we will study in more detail in Chapter 6). In most cases, the methods are designed based on certain assumptions about the data distributions and tailored to specific problems.

More recently, deep neural networks have achieved remarkable success across a wide range of data and applications. A neural network

$$f(\cdot, \boldsymbol{\theta}) \colon \boldsymbol{x} \xrightarrow{f^{\text{pre}}} \boldsymbol{z}^1 \to \cdots \to \boldsymbol{z}^\ell \xrightarrow{f^\ell} \boldsymbol{z}^{\ell+1} \to \cdots \to \boldsymbol{z}^{L+1} = \boldsymbol{z}. \tag{5.0.1}$$

can learn effective features/representations for downstream applications. For example, a trained deep neural network $f(\cdot, \boldsymbol{\theta})$ can be applied to map images to feature vectors, that is, $\boldsymbol{z}_i = f(\boldsymbol{x}_i, \boldsymbol{\theta})$, while a linear classifier can be learned on top of such representations $\{\boldsymbol{z}_i\}$. One notable breakthrough is AlexNet [KSH12], a deep convolutional neural network trained with more than a million natural images, outperforming (on predictive tasks) all previous approaches that were based on hand-crafted features. One of the key differences between AlexNet and previous approaches is that the former *learns parameters of the nonlinear transformation from massive amounts of data* trained with back-propagation (BP) [RHW86b], as detailed in Section A.2.3 of Chapter A.

Subsequent popular practice models the mapping f with other empirically designed artificial deep neural networks and learns the parameters $\boldsymbol{\theta}$ from random initialization via BP. Starting with the AlexNet [KSH12], the architectures of modern deep networks continue to be empirically revised and improved. Network architectures such as VGG [SZ14], ResNet [HZR+16b], DenseNet [HLV+17], CNN, RNN or LSTM [HS97], Transformer [VSP+17b], and mixtures of experts (MoE) [FZS22; SMM+17], etc. have continued to push the performance envelope. As part of the effort to improve the performance of deep networks, almost every component of the networks has been empirically scrutinized, and various revisions and improvements have been proposed. They are not limited to nonlinear activation functions [KUM+17; MHN13; NIG+18; XWC+15], skip connections [HZR+16b; RFB15b], normalizations [BKH16; IS15; MKK+18; UVL16; WH20], up/down sampling or pooling [SMB10], convolutions [KSH12; LBB+98b], etc. However, almost all such modifications have been developed through years of empirical trial and error or ablation studies. Some recent practices even take to the extreme by searching for effective network structures and training strategies through extensive random search techniques, such as Neural Architecture Search [BGN+17; ZL17], AutoML [HKV19], and Learning to Learn [ADG+16].

Despite the wide application of deep neural networks, it is not clear what the underlying design principles of such a constructed network are. In particular, it is not clear what mathematical function each layer of the network performs. In this chapter, based on the results from previous chapters, we develop a principled framework that will provide a fully rigorous mathematical interpretation of the role of a deep network, including its individual layers and the network as a whole.

To understand deep networks and how they should be better designed, we must start with the objective of representation learning. In previous chapters, we have argued that the objective is to identify the intrinsically low-dimensional data distribution and then transform it to a compact and structured (say piecewise linear) representation. As we have seen in the previous chapter, the general approach to identifying a low-dimensional data distribution is through a compression process that progressively minimizes the entropy or coding rate of the distribution. However, up to this point, we have been using empirically designed deep networks to model or approximate the operations that aim to optimize these objectives, such as the score function for denoising (in Section 1.3.1) or

the transformation that maximizes the rate reduction (in Section 4.2.4).

As we have argued in the previous chapter, Section 4.2 in particular, one can measure the goodness of the resulting representation by the information of the representation gained from a "lazy" representation which models all data as one big Gaussian.[1] In particular, if *we use a mixture of Gaussians (subspaces)*[2] *as prototypical distributions to approximate the non-linear distribution of interest*, then we can efficiently measure the coding rate of such a representation using the (sum of) rate distortion functions of the associated Gaussians. Then the amount of *information gained* or (relative) entropy reduced with such a modeling can be measured by the difference between the coding rate for the lazy representation and that for the more refined representation. Then, the objective of representation learning is to maximize this information gain, also known as the rate reduction objective.

As we will see in this chapter, once the objective of representation learning is clear, the role of a deep neural network is precisely to help optimize the objective iteratively. Each layer of a deep neural network can be naturally derived as an iterative optimization step to incrementally maximize the information gain, including the popular architectures of ResNet, CNN, and Transformer, and other more advanced variants. In particular, this chapter aims to answer the following questions about deep networks:

- Section 5.1 — given a measure of goodness for a learned representation, how to construct the nonlinear mapping from the data to the optimal representation via unrolled optimization for the objective?
- Section 5.2 — how would the above unrolling approach provide a principled interpretation of the popular transformer architectures; if so, what are the associated objective and optimization mechanisms?
- Section 5.3 — how would this framework guide us to design more efficient or more parsimonious deep architectures?

5.1 White-Box Deep Networks via Unrolled Optimization

Now, if we agree that maximizing the rate reduction or information gain leads to the desired representation as discussed in Section 4.2, the remaining question is how to construct and learn a (nonlinear) mapping from the data $\boldsymbol{X}$ to the optimal representation $\boldsymbol{Z}^*$. This involves designing a network architecture and learning algorithm that can effectively capture the underlying structures in the data and faithfully realize the optimal representation.

[1] that we have seen in the previous chapter as one particular choice of interpretation of the sampled dataset.

[2] which we have studied thoroughly in the previous chapter.

5.1.1 Deep Networks from Unrolled Gradient Descent

In the previous chapter, we presented the rate reduction objective (4.2.15) as a principled objective for learning linear discriminative representations of the data. We have, however, not specified the architecture of the feature mapping $\boldsymbol{z} = f(\boldsymbol{x}, \boldsymbol{\theta})$ for extracting such representations from input data $\boldsymbol{x}$. A straightforward choice is to use a conventional deep network, such as ResNet, for implementing $f(\boldsymbol{x}, \boldsymbol{\theta})$. As we have seen in Section 4.3.1, such a choice often leads to decent performance empirically. Nonetheless, there remain several unanswered problems with adopting an arbitrary deep network. Although the learned feature representation is now more interpretable, the network itself is still *not*. It is unclear why any chosen "black-box" network is able to optimize the desired MCR^2 objective at all. The good empirical results (say with a ResNet) do not necessarily justify the particular choice in architectures and operators of the network: Why is a deep layered model even necessary; what do additional layers try to improve or simplify; how wide and deep is adequate; or is there any rigorous justification for the convolutions (in a popular multi-channel form) and nonlinear operators (e.g. ReLU or softmax) used?

In this chapter, we show that using gradient ascent to maximize the rate reduction $\Delta R_\epsilon(\boldsymbol{Z} \mid \boldsymbol{\Pi})$ as defined in (4.2.15) naturally leads to a "white-box" deep network that realizes the desired mapping. All network layers, linear/nonlinear operators, and parameters are *explicitly constructed in a purely forward propagation fashion*. Moreover, such network architectures resemble existing empirically-designed deep networks, providing principled justifications for their design.

Gradient ascent for coding rate reduction. From the previous chapter, we see that to seek a linear discriminative representation (LDR), mathematically, we are essentially seeking a continuous mapping $f(\cdot) : \boldsymbol{x} \mapsto \boldsymbol{z}$ from the data $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N] \in \mathbb{R}^{D \times N}$ (or initial features extracted from the data[3]) to an optimal representation $\boldsymbol{Z} = [\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N] \in \mathbb{R}^{d \times N}$ that maximizes the following coding rate reduction objective (also see (4.2.17)):

$$\Delta R_\epsilon(\boldsymbol{Z} \mid \boldsymbol{\Pi}) \doteq \underbrace{\frac{1}{2} \log\det\Big(\boldsymbol{I} + \alpha \boldsymbol{Z}\boldsymbol{Z}^\top\Big)}_{R_\epsilon(\boldsymbol{Z})} - \underbrace{\sum_{k=1}^{K} \frac{\gamma_k}{2} \log\det\Big(\boldsymbol{I} + \alpha_k \boldsymbol{Z}\boldsymbol{\Pi}_k\boldsymbol{Z}^\top\Big)}_{R_\epsilon^c(\boldsymbol{Z}\mid\boldsymbol{\Pi})}, \tag{5.1.1}$$

where $\epsilon > 0$ is a prescribed quantization error and for simplicity we denote[4]

$$\alpha \doteq \frac{d}{N\epsilon^2}, \quad \alpha_k \doteq \frac{d}{\operatorname{tr}(\boldsymbol{\Pi}_k)\epsilon^2}, \quad \gamma_k \doteq \frac{\operatorname{tr}(\boldsymbol{\Pi}_k)}{N}, \quad \text{for } k = 1, \ldots, K. \tag{5.1.2}$$

The question really boils down to whether there is a *constructive* way of finding such a continuous mapping $f(\cdot, \boldsymbol{\theta})$ from $\boldsymbol{x}$ to $\boldsymbol{z}$? To this end, let us consider incrementally maximizing the objective ΔR_ϵ as a function of $\boldsymbol{Z} \subseteq$

[3] As we will see the necessity of such a feature extraction in the next section.

[4] Notice our use of slightly simplified notation compared to Chapter 3.

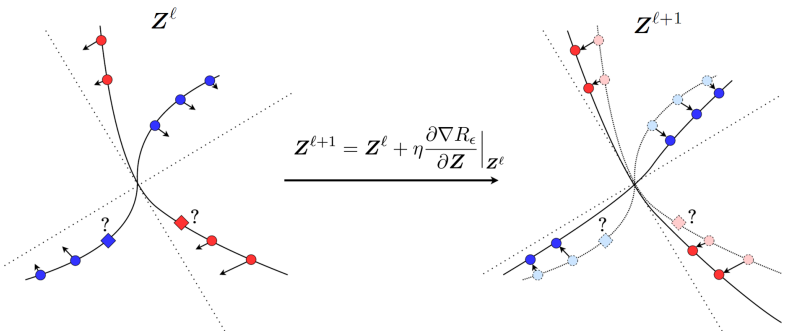


Figure 5.1: Incremental deformation via gradient flow to both flatten data of each class into a subspace and push different classes apart.

$\mathbb{S}^{d-1}$. Although there might be many optimization schemes to choose from, for simplicity we first consider the arguably simplest projected *gradient ascent* (PGA) scheme:[5]

$$\boldsymbol{Z}^{\ell+1} \propto \boldsymbol{Z}^\ell + \eta \cdot \frac{\partial \Delta R_\epsilon}{\partial \boldsymbol{Z}}(\boldsymbol{Z}^\ell) \quad \text{s.t.} \quad \boldsymbol{Z}^{\ell+1} \subseteq \mathbb{S}^{d-1}, \quad \ell = 1, 2, \ldots, \tag{5.1.3}$$

for some step size $\eta > 0$ and the iterate starts with the given data $\boldsymbol{Z}^0 = \boldsymbol{X}$.[6] This scheme can be interpreted as how one should incrementally adjust locations of the current features $\boldsymbol{Z}^\ell$, initialized as the input data $\boldsymbol{X}$, in order for the resulting $\boldsymbol{Z}^{\ell+1}$ to improve the rate reduction ΔR_ϵ, as illustrated in Figure 5.1.

Simple calculation shows that the gradient $\partial \Delta R_\epsilon / \partial \boldsymbol{Z}$ entails evaluating the following derivatives of the two terms in ΔR_ϵ:

$$\frac{1}{2}\frac{\partial \log\det(\boldsymbol{I}+\alpha \boldsymbol{Z}\boldsymbol{Z}^\top)}{\partial \boldsymbol{Z}}(\boldsymbol{Z}^\ell) = \underbrace{\alpha(\boldsymbol{I}+\alpha \boldsymbol{Z}^\ell(\boldsymbol{Z}^\ell)^\top)^{-1}}_{\boldsymbol{E}^\ell \in \mathbb{R}^{d\times d}} \boldsymbol{Z}^\ell, \tag{5.1.4}$$

$$\frac{1}{2}\frac{\partial \left(\gamma_k \log\det(\boldsymbol{I} + \alpha_k \boldsymbol{Z}\boldsymbol{\Pi}_k \boldsymbol{Z}^\top)\right)}{\partial \boldsymbol{Z}}(\boldsymbol{Z}^\ell) = \gamma_k \underbrace{\alpha_k(\boldsymbol{I} + \alpha_k \boldsymbol{Z}^\ell \boldsymbol{\Pi}_k (\boldsymbol{Z}^\ell)^\top)^{-1}}_{\boldsymbol{C}_k^\ell \in \mathbb{R}^{d\times d}} \boldsymbol{Z}^\ell \boldsymbol{\Pi}_k. \tag{5.1.5}$$

Notice that in the above, the matrix $\boldsymbol{E}^\ell$ only depends on $\boldsymbol{Z}^\ell$ and it aims to *expand* all the features to increase the overall coding rate; the matrix $\boldsymbol{C}_k^\ell$ depends

[5] Notice that we use subscript j on $\boldsymbol{Z}_j$ to indicate features in the j-th class and superscript ℓ on $\boldsymbol{Z}^\ell$ to indicate all features at ℓ-th iteration or layer.

[6] Again, for simplicity, we here first assume the initial features $\boldsymbol{Z}^1$ are the data themselves. Note that here ℓ denotes the number of iterations. Hence, the data and the features have the same dimension d. This needs not to be the case though. As we will see in the next section, the initial features can be some (lifted) features of the data to begin with and could in principle have a different (much higher) dimension. All subsequent iterates have the same dimension.

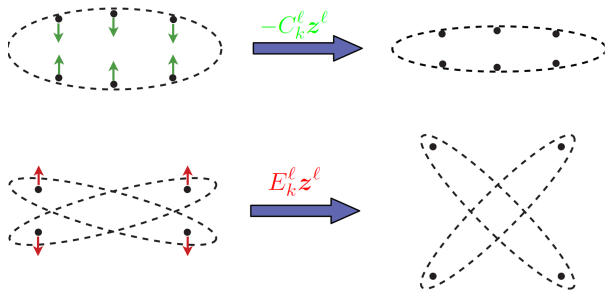


Figure 5.2: Interpretation of $\boldsymbol{C}_k^\ell$ and $\boldsymbol{E}^\ell$: $\boldsymbol{C}_k^\ell$ compresses each class by contracting the features to a low-dimensional subspace; $\boldsymbol{E}^\ell$ expands all features by contrasting and repelling features across different classes.

on features from the k-class and aims to *compress* them to reduce the coding rate of each class. Then the complete gradient $\frac{\partial \Delta R_\epsilon}{\partial \boldsymbol{Z}}(\boldsymbol{Z}^\ell) \in \mathbb{R}^{d\times N}$ is of the form:

$$\frac{\partial \Delta R_\epsilon}{\partial \boldsymbol{Z}}(\boldsymbol{Z}^\ell) = \underbrace{\boldsymbol{E}^\ell}_{\text{Expansion}} \boldsymbol{Z}^\ell - \sum_{k=1}^{K} \gamma_k \underbrace{\boldsymbol{C}_k^\ell}_{\text{Compression}} \boldsymbol{Z}^\ell \boldsymbol{\Pi}_k. \tag{5.1.6}$$

Remark 5.1 (Interpretation of $\boldsymbol{E}^\ell$ and $\boldsymbol{C}_j^\ell$ as linear operators). For any $\boldsymbol{z}^\ell \in \mathbb{R}^d$,

$$\boldsymbol{E}^\ell \boldsymbol{z}^\ell = \alpha(\boldsymbol{z}^\ell - \boldsymbol{Z}^\ell \boldsymbol{q}_\star^\ell), \quad \text{where} \quad \boldsymbol{q}_\star^\ell \doteq \arg\min_{\boldsymbol{q}^\ell} \left\{\alpha\|\boldsymbol{z}^\ell - \boldsymbol{Z}^\ell \boldsymbol{q}^\ell\|_2^2 + \|\boldsymbol{q}^\ell\|_2^2\right\}. \tag{5.1.7}$$

Notice that $\boldsymbol{q}_\star^\ell$ is exactly the solution to the ridge regression by all the data points $\boldsymbol{Z}^\ell$ concerned. Therefore, $\boldsymbol{E}^\ell$ (similarly for $\boldsymbol{C}_k^\ell$) is approximately (i.e., when N is large enough) the projection onto the orthogonal complement of the subspace spanned by columns of $\boldsymbol{Z}^\ell$. Another way to interpret the matrix $\boldsymbol{E}^\ell$ is through eigenvalue decomposition of the covariance matrix $\boldsymbol{Z}^\ell(\boldsymbol{Z}^\ell)^\top$. Assuming that $\boldsymbol{Z}^\ell(\boldsymbol{Z}^\ell)^\top \doteq \boldsymbol{U}^\ell \boldsymbol{\Lambda}^\ell (\boldsymbol{U}^\ell)^\top$ where $\boldsymbol{\Lambda}^\ell \doteq \operatorname{diag}\left(\lambda_1^\ell, \ldots, \lambda_d^\ell\right)$, we have

$$\boldsymbol{E}^\ell = \alpha \boldsymbol{U}^\ell \operatorname{diag}\left(\frac{1}{1+\alpha\lambda_1^\ell}, \ldots, \frac{1}{1+\alpha\lambda_d^\ell}\right)\left(\boldsymbol{U}^\ell\right)^\top. \tag{5.1.8}$$

Therefore, the matrix $\boldsymbol{E}^\ell$ operates on a vector $\boldsymbol{z}^\ell$ by stretching in a way that directions of large variance are shrunk while directions of vanishing variance are kept. These are exactly the directions (5.1.4) in which we move the features so that the overall volume expands and the coding rate will increase, hence the positive sign. To the opposite effect, the directions associated with (5.1.5) are "residuals" of features of each class that deviate from the subspace to which they are supposed to belong. These are exactly the directions in which the features need to be compressed back onto their respective subspace, hence the negative sign (see Figure 5.2).

Essentially, the linear operations $\boldsymbol{E}^\ell$ and $\boldsymbol{C}_k^\ell$ in gradient ascent for rate reduction are determined by training data conducting "auto-regressions". The recent renewed understanding about ridge regression in an over-parameterized setting [WX20; YYY+20] indicates that using seemingly redundantly sampled data (from each subspace) as regressors do not lead to overfitting.

Gradient-guided feature map increment. Notice that in the above, the gradient ascent considers all the features $\boldsymbol{Z}^\ell = [\boldsymbol{z}_1^\ell, \ldots, \boldsymbol{z}_N^\ell]$ as free variables. The increment $\boldsymbol{Z}^{\ell+1} - \boldsymbol{Z}^\ell = \eta \frac{\partial \Delta R_\epsilon}{\partial \boldsymbol{Z}}(\boldsymbol{Z}^\ell)$ does not yet give a transformation on the entire feature domain $\boldsymbol{z}^\ell \in \mathbb{R}^d$. According to equation (5.1.6), the gradient cannot be evaluated at a point whose membership is not known, as illustrated in Figure 5.1. Hence, in order to find the optimal $f(\boldsymbol{x}, \boldsymbol{\theta})$ explicitly, we may consider constructing a small increment transform $g(\cdot, \theta^\ell)$ on the ℓ-th layer feature $\boldsymbol{z}^\ell$ to emulate the above (projected) gradient scheme:

$$\boldsymbol{z}^{\ell+1} \;\propto\; \boldsymbol{z}^\ell + \eta \cdot g(\boldsymbol{z}^\ell, \theta^\ell) \quad \text{subject to} \quad \boldsymbol{z}^{\ell+1} \in \mathbb{S}^{d-1} \tag{5.1.9}$$

such that $\big[g(\boldsymbol{z}_1^\ell, \boldsymbol{\theta}^\ell), \ldots, g(\boldsymbol{z}_N^\ell, \boldsymbol{\theta}^\ell)\big] \approx \frac{\partial \Delta R_\epsilon}{\partial \boldsymbol{Z}}(\boldsymbol{Z}^\ell)$. That is, we need to approximate the gradient flow $\frac{\partial \Delta R_\epsilon}{\partial \boldsymbol{Z}}$ that locally deforms all (training) features $\{\boldsymbol{z}_i^\ell\}_{i=1}^N$ with a continuous mapping $g(\boldsymbol{z}, \boldsymbol{\theta})$ defined on the entire feature space $\boldsymbol{z}^\ell \in \mathbb{R}^d$. Notice that one may interpret the increment (5.1.9) as a discretized version of a continuous differential equation:

$$\dot{\boldsymbol{z}} = g(\boldsymbol{z}, \theta). \tag{5.1.10}$$

Hence the (deep) network so constructed can be interpreted as a certain neural ODE [CRB+18]. Nevertheless, unlike neural ODE where the flow g is chosen to be some generic structures, here our $g(\boldsymbol{z}, \boldsymbol{\theta})$ is to emulate the gradient flow of the rate reduction on the feature set (as shown in Figure 5.1):

$$\dot{\boldsymbol{Z}} = \frac{\partial \Delta R_\epsilon}{\partial \boldsymbol{Z}},$$

and its structure is entirely derived and fully determined from this objective, without any other priors or heuristics.

By inspecting the structure of the gradient (5.1.6), it suggests that a natural candidate for the increment transform $g(\boldsymbol{z}^\ell, \boldsymbol{\theta}^\ell)$ is of the form:

$$g(\boldsymbol{z}^\ell, \boldsymbol{\theta}^\ell) \;\doteq\; \boldsymbol{E}^\ell \boldsymbol{z}^\ell - \sum_{k=1}^K \gamma_k \pi_k(\boldsymbol{z}^\ell) \boldsymbol{C}_k^\ell \boldsymbol{z}^\ell \in \mathbb{R}^d, \tag{5.1.11}$$

where $\pi_k(\boldsymbol{z}^\ell) \in [0, 1]$ indicates the probability of $\boldsymbol{z}^\ell$ belonging to the k-th class. The increment map parameters $\boldsymbol{\theta}^\ell$ depend on: First, a set of linear maps represented by $\boldsymbol{E}^\ell$ and $\{\boldsymbol{C}_k^\ell\}_{k=1}^K$ that depend only on statistics of features of the training $\boldsymbol{Z}^\ell$; Second, the membership $\{\pi_k(\boldsymbol{z}^\ell)\}_{k=1}^K$ of any feature $\boldsymbol{z}^\ell$. Notice that on the training samples $\boldsymbol{Z}^\ell$, for which the memberships $\boldsymbol{\Pi}_k$ are known, the so defined $g(\boldsymbol{z}^\ell, \boldsymbol{\theta})$ gives exactly the values for the gradient $\frac{\partial \Delta R_\epsilon}{\partial \boldsymbol{Z}}(\boldsymbol{Z}^\ell)$.

Since we only have the membership for the training samples, the function $g(\cdot)$ defined in (5.1.11) can only be evaluated on the training. To extrapolate $g(\cdot)$ to the entire feature space, we need to estimate $\pi_k(\boldsymbol{z}^\ell)$ in its second term. In conventional deep learning, this map is typically modeled as a deep network and learned from the training data, say via *back propagation*. Nevertheless, our goal here is not to learn a precise classifier $\pi_k(\boldsymbol{z}^\ell)$ already. Instead, we only need a good enough estimate of the class information in order for $g(\cdot)$ to approximate the gradient $\frac{\partial \Delta R_\epsilon}{\partial \boldsymbol{Z}}$ well.

From the geometric interpretation of the linear maps $\boldsymbol{E}^\ell$ and $\boldsymbol{C}_k^\ell$ given by Remark 5.1, the term $\boldsymbol{p}_k^\ell \doteq \boldsymbol{C}_k^\ell \boldsymbol{z}^\ell$ can be viewed as (approximately) the projection of $\boldsymbol{z}^\ell$ onto the orthogonal complement of each class j. Therefore, $\|\boldsymbol{p}_j^\ell\|_2$ is small if $\boldsymbol{z}^\ell$ is in class j and large otherwise. This motivates us to estimate its membership based on the following softmax function:

$$\widehat{\boldsymbol{\pi}}(\boldsymbol{z}^\ell) \doteq \operatorname{softmax}\left(-\lambda \begin{bmatrix} \|\boldsymbol{C}_1^\ell \boldsymbol{z}^\ell\|_2 \\ \vdots \\ \|\boldsymbol{C}_K^\ell \boldsymbol{z}^\ell\|_2 \end{bmatrix}\right) \tag{5.1.12}$$

$$= \frac{1}{\sum_{k=1}^K \exp(-\lambda \|\boldsymbol{C}_k^\ell \boldsymbol{z}^\ell\|_2)} \begin{bmatrix} \exp(-\lambda \|\boldsymbol{C}_1^\ell \boldsymbol{z}^\ell\|_2) \\ \vdots \\ \exp(-\lambda \|\boldsymbol{C}_K^\ell \boldsymbol{z}^\ell\|_2) \end{bmatrix} \in [0,1]^K. \tag{5.1.13}$$

Hence, the second term of (5.1.11) can be approximated by this estimated membership:

$$\sum_{k=1}^K \gamma_k \pi_k(\boldsymbol{z}^\ell) \boldsymbol{C}_k^\ell \boldsymbol{z}^\ell \;\approx\; \sum_{k=1}^K \gamma_k \widehat{\pi}_k(\boldsymbol{z}^\ell) \boldsymbol{C}_k^\ell \boldsymbol{z}^\ell \;\doteq\; \boldsymbol{\sigma}\Big([\boldsymbol{C}_1^\ell \boldsymbol{z}^\ell, \ldots, \boldsymbol{C}_K^\ell \boldsymbol{z}^\ell]\Big), \tag{5.1.14}$$

which is denoted as a nonlinear operator $\boldsymbol{\sigma}(\cdot)$ on outputs of the feature $\boldsymbol{z}^\ell$ through K groups of filters: $[\boldsymbol{C}_1^\ell, \ldots, \boldsymbol{C}_K^\ell]$. Notice that the nonlinearality arises due to a "soft" assignment of class membership based on the feature responses from those filters.

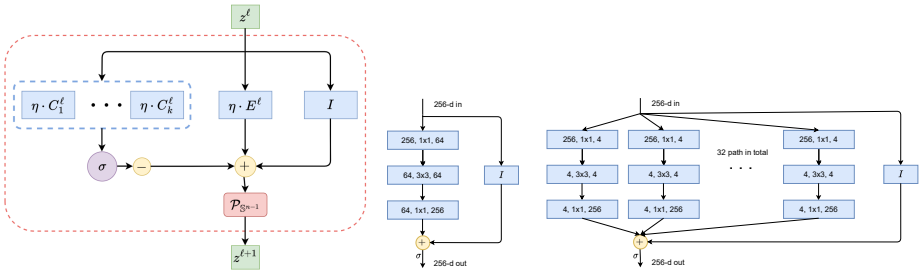
Overall, combining (5.1.9), (5.1.11), and (5.1.14), the increment feature transform from $\boldsymbol{z}^\ell$ to $\boldsymbol{z}^{\ell+1}$ now becomes

$$\boldsymbol{z}^{\ell+1} \propto \boldsymbol{z}^\ell + \eta \cdot \boldsymbol{E}^\ell \boldsymbol{z}^\ell - \eta \cdot \boldsymbol{\sigma}\big([\boldsymbol{C}_1^\ell \boldsymbol{z}^\ell, \ldots, \boldsymbol{C}_K^\ell \boldsymbol{z}^\ell]\big) \tag{5.1.15}$$

$$= \boldsymbol{z}^\ell + \eta \cdot g(\boldsymbol{z}^\ell, \boldsymbol{\theta}^\ell) \qquad \text{s.t.} \quad \boldsymbol{z}^{\ell+1} \in \mathbb{S}^{d-1}, \tag{5.1.16}$$

with the nonlinear function $\boldsymbol{\sigma}(\cdot)$ defined above and $\boldsymbol{\theta}^\ell$ collecting all the layer-wise parameters. That is $\boldsymbol{\theta}^\ell = \{\boldsymbol{E}^\ell, \boldsymbol{C}_1^\ell, \ldots, \boldsymbol{C}_K^\ell, \gamma_k, \lambda\}$. Note features at each layer are always "normalized" by projecting onto the unit sphere $\mathbb{S}^{d-1}$, denoted as $\mathcal{P}_{\mathbb{S}^{d-1}}$, i.e., dividing each feature by its norm.[7] The form of increment in (5.1.15) can be illustrated by a diagram in Figure 5.3(a).

[7] This mirrors the common LayerNorm block in neural networks, which computes the projected feature and then multiplies and adds by trainable parameters [BKH16], see Section 8.2.3 for more details.



(a) **ReduNet** (b) **ResNet** and **ResNeXt**.

Figure 5.3: Network Architectures of the ReduNet and comparison with others. **(a)**: Layer structure of the **ReduNet** derived from one iteration of gradient ascent for optimizing rate reduction. **(b)** (left): A layer of ResNet [HZR+16b]; and **(b)** (right): A layer of ResNeXt [XGD+17]. As we will see in Section 5.1.2, the linear operators $\boldsymbol{E}^\ell$ and $\boldsymbol{C}_k^\ell$ of the ReduNet naturally become (multi-channel) convolutions when shift-invariance is imposed.

Deep network for optimizing rate reduction. Notice that the increment is constructed to emulate the gradient ascent for the rate reduction ΔR_ϵ. Hence by transforming the features iteratively via the above process, we expect the rate reduction to increase, as we will see in the experimental section. This iterative process, once converged, say after L iterations, gives the desired feature map $f(\boldsymbol{x}, \boldsymbol{\theta})$ on the input $\boldsymbol{x} = \boldsymbol{z}^0$, precisely in the form of a *deep network*, in which each layer has the structure shown in Figure 5.3 left:

$$f(\boldsymbol{x}, \boldsymbol{\theta}) \;=\; f^L \circ f^{L-1} \circ \cdots \circ f^1 \circ f^0(\boldsymbol{z}^0), \tag{5.1.17}$$

$$f^\ell(\boldsymbol{z}^\ell, \boldsymbol{\theta}^\ell) \;\doteq\; \boldsymbol{z}^{\ell+1} = \mathcal{P}_{\mathbb{S}^{n-1}}[\boldsymbol{z}^\ell + \eta \cdot g(\boldsymbol{z}^\ell, \boldsymbol{\theta}^\ell)], \tag{5.1.18}$$

$$g(\boldsymbol{z}^\ell, \boldsymbol{\theta}^\ell) \;=\; \boldsymbol{E}^\ell \boldsymbol{z}^\ell - \boldsymbol{\sigma}\big([\boldsymbol{C}_1^\ell \boldsymbol{z}^\ell, \ldots, \boldsymbol{C}_K^\ell \boldsymbol{z}^\ell]\big). \tag{5.1.19}$$

As this deep network is derived from maximizing the rate **redu**ction, we call it the **ReduNet**. By comparing the architecture of ReduNet with those of popular empirically designed networks, **ResNet** and **ResNeXt** shown in Figure 5.3, the similarity is somewhat uncanny. Conceptually, ReduNet could also be used to justify the popular mixture of experts (**MoE**) architecture [SMM+17] as each parallel channel, $\boldsymbol{C}_k^\ell$, can be viewed as an "expert" trained for each class of objects.

We summarize the training and evaluation of ReduNet in Algorithm 5.1 and Algorithm 5.2, respectively. Notice that all parameters of the network are explicitly constructed layer by layer in a *forward propagation* fashion. The construction does not need any back propagation! The so-learned features can be directly used for classification, say via a nearest subspace classifier.

Example 5.1. To provide some intuition on how ReduNet transforms the features, we provide a simple example with mixed 3D Gaussians and visualize how the features are transformed in Figure 5.5. Consider a mixture of three Gaussian distributions in $\mathbb{R}^3$ that is projected onto $\mathbb{S}^2$. We first generate data points

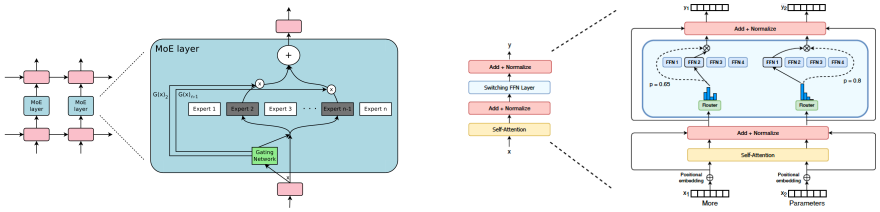


Figure 5.4: Left: a mixture of experts (MoE) deep network [SMM+17]. Right: a sparsity-promoting Switch Transformer [FZS22], used to implement MoE with 1.7 trillion parameters.

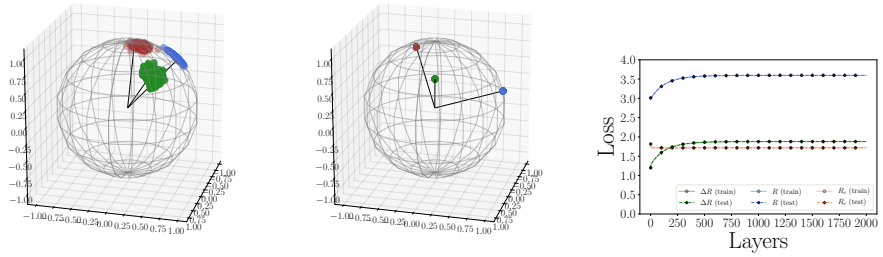
for 3 classes: for $k = 1, 2, 3$, $\boldsymbol{X}_k = [\boldsymbol{x}_{k,1}, \ldots, \boldsymbol{x}_{k,m}] \in \mathbb{R}^{3\times m}$, $\boldsymbol{x}_{k,i} \sim \mathcal{N}(\boldsymbol{\mu}_k, \sigma_k^2 \boldsymbol{I})$, and $\pi(\boldsymbol{x}_{k,i}) = k$. We set $m = 500, \sigma_1 = \sigma_2 = \sigma_3 = 0.1$, and $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\mu}_3 \in \mathbb{S}^2$. Then we project all the data points onto $\mathbb{S}^2$, i.e., $\boldsymbol{x}_{k,i}/\|\boldsymbol{x}_{k,i}\|_2$. To construct the network (computing $\boldsymbol{E}^\ell, \boldsymbol{C}_k^\ell$ for the ℓ-th layer), we set the number of iterations/layers $L = 2,000$, step size $\eta = 0.5$, and precision $\epsilon = 0.1$. We do this only to demonstrate that our framework leads to stable deep networks even with thousands of layers. In practice, thousands of layers may not be necessary and one can stop whenever adding new layers gives diminishing returns. For this example, a couple of hundred layers is sufficient. Hence, the clear optimization objective gives a natural criterion for the depth of the network needed.

As shown in Figure 5.5, we can observe that after the mapping $f(\cdot, \theta)$, samples from the same class are highly compressed and converge to a single cluster and the angle between two different clusters is approximately $\pi/2$, which is well aligned with the optimal solution $\boldsymbol{Z}^\star$ of the MCR^2 loss in $\mathbb{S}^2$. MCR^2 loss of features on different layers can be found in Figure 5.5(**c**). Empirically, we find that the constructed ReduNet is able to maximize MCR^2 loss and converges stably and samples from the same class converge to one cluster and points in different clusters are orthogonal to each other.[8] Moreover, when sampling new data points from the same distributions, we find that new samples from the same class consistently converge to the same cluster center as the training samples. ■

5.1.2 Convolutional Networks from Invariant Rate Reduction

In the previous section, we derived the layer-wise architecture of a deep network, the ReduNet, using unrolled optimization for the rate reduction objective. Specifically, the compression term $R_\epsilon^c(\boldsymbol{Z} \mid \boldsymbol{\Pi})$ in (5.1.1) is designed to compress representations from the same class. However, this formulation does not account for possible domain transformation or deformation of the input data. For

[8] Because of the low dimension, each cluster collapses to a 1-dimensional subspace on the sphere (e.g., a single point and its antipode); this is the extreme case of keeping within-cluster dimension minimal. In higher dimensions, of course, subspaces will be less compressed.



(a) $\boldsymbol{X}_{\text{train}}$ (b) $\boldsymbol{Z}_{\text{train}}$ (c) Loss (train/val)

Figure 5.5: Original samples and learned representations for 3D Mixture of Gaussians. We visualize data points $\boldsymbol{X}$ (before mapping $f(\cdot, \theta)$) in (**a**) and learned features $\boldsymbol{Z}$ (after mapping $f(\cdot, \theta)$) in (**b**) by scatter plot. In each scatter plot, each color represents one class of samples. In (**c**), we also show the plots for the progression of values of the objective functions.

instance, shifting an object slightly to the right does not change the semantic label of an image. In this section, we will demonstrate how convolutional layers can be derived by maximizing a rate reduction objective that is invariant to certain domain deformations, such as image rotations and translations.

For many clustering or classification tasks (such as object detection in images), we consider two samples as *equivalent* if they differ by certain classes of domain deformations or augmentations $\mathcal{T} = \{\tau\}$. Hence, we are only interested in low-dimensional structures that are *invariant* to such deformations (i.e., $\boldsymbol{x} \in \mathcal{M}$ iff $\tau(\boldsymbol{x}) \in \mathcal{M}$ for all $\tau \in \mathcal{T}$), which are known to have sophisticated geometric and topological structures and can be difficult to learn precisely in practice, even with rigorously designed CNNs [CW16a]. In this framework, this can be formulated in a very natural way: all equivariant instances are to be embedded into the same subspace, so that the subspace itself is invariant to the transformations under consideration.

In many applications, such as serial data or imagery data, the semantic meaning (labels) of the data are *invariant* to certain transformations $\mathfrak{g} \in \mathbb{G}$ (for some group $\mathbb{G}$) [CW16c; ZKR+17]. For example, the meaning of an audio signal is invariant to shift in time; and the identity of an object in an image is invariant to translation in the image plane. Hence, we prefer the feature mapping $f(\boldsymbol{x}, \boldsymbol{\theta})$ is rigorously invariant to such transformations:

$$\textit{Group Invariance: } f(\boldsymbol{x} \circ \mathfrak{g}, \boldsymbol{\theta}) \sim f(\boldsymbol{x}, \boldsymbol{\theta}), \ \forall \mathfrak{g} \in \mathbb{G}, \tag{5.1.20}$$

where "$\sim$" indicates two features belonging to the same equivalent class. Although to ensure invariance or equivarience, convolutional operators have been common practice in deep networks [CW16c], it remains challenging in practice to train an (empirically designed) convolution network from scratch that can *guarantee* invariance even to simple transformations such as translation and rotation [AW18; ETT+17]. An alternative approach is to carefully design convolution filters of each layer so as to ensure translational invariance for a wide range

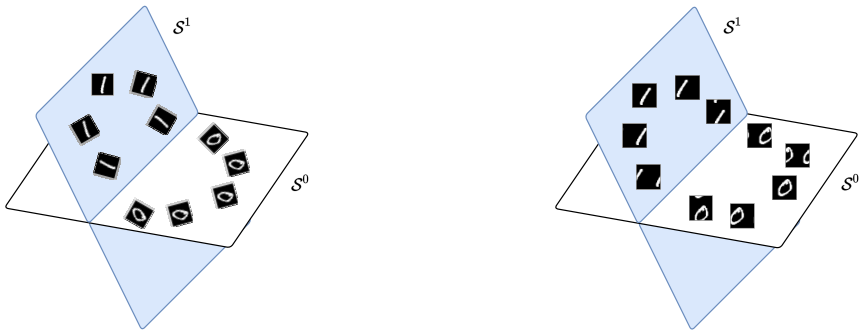


Figure 5.6: Illustration of the sought representation that is equivariant/invariant to image rotation (left) or translation (right): all transformed images of each class are mapped into the same subspace that is incoherent to other subspaces. The features embedded in each subspace are equivariant to the transformation group whereas each subspace is invariant to such transformations.

of signals, say using wavelets as in ScatteringNet [BM13] and followup works [WB18]. However, in order to ensure invariance to generic signals, the number of convolutions needed usually grows exponentially with network depth. That is the reason why this type of network cannot be constructed so deep, usually only several layers.

Now, we show that the MCR^2 principle is compatible with invariance in a natural and precise way: we only need to assign all transformed versions $\{\boldsymbol{x} \circ \mathfrak{g} \mid \mathfrak{g} \in \mathbb{G}\}$ into the same class as the data $\boldsymbol{x}$ and map their features $\boldsymbol{z}$ all to the same subspace $\mathcal{S}$. Hence, all group equivariant information is encoded only inside the subspace, and any classifier defined on the resulting set of subspaces will be automatically invariant to such group transformations. See Figure 5.6 for an illustration of the examples of 1D rotation and 2D translation. Next, we will rigorously show that when the group $\mathbb{G}$ is circular 1D shifting, the resulting deep network naturally becomes a *multi-channel convolution network*. Because the so-constructed network only needs to ensure invariance for the given data $\boldsymbol{X}$ or their features $\boldsymbol{Z}$, the number of convolutions needed actually remains constant through a very deep network, as opposed to the ScatteringNet.

1D serial data and shift invariance To classify one-dimensional data $\boldsymbol{x} = [x(0), x(1), \ldots, x(D-1)] \in \mathbb{R}^D$ invariant under shifting, we take $\mathbb{G}$ to be the group of all circular shifts. Each observation $\boldsymbol{x}_i$ generates a family $\{\boldsymbol{x}_i \circ \mathfrak{g} \,|\, \mathfrak{g} \in \mathbb{G}\}$ of shifted copies, which are the columns of the circulant matrix $\mathsf{circ}(\boldsymbol{x}_i) \in \mathbb{R}^{D \times D}$

given by

$$\mathsf{circ}(\boldsymbol{x}) \doteq \begin{bmatrix} x(0) & x(D-1) & \dots & x(2) & x(1) \\ x(1) & x(0) & x(D-1) & \cdots & x(2) \\ \vdots & x(1) & x(0) & \ddots & \vdots \\ x(D-2) & \vdots & \ddots & \ddots & x(D-1) \\ x(D-1) & x(D-2) & \dots & x(1) & x(0) \end{bmatrix} \in \mathbb{R}^{D\times D}. \tag{5.1.21}$$

We refer the reader to [KS12] for properties of circulant matrices. For simplicity, let $\boldsymbol{Z}^1 \doteq [\boldsymbol{z}_1^1, \dots, \boldsymbol{z}_N^1] = \boldsymbol{X} \in \mathbb{R}^{d\times N}$.[9] Then what happens if we construct the ReduNet from their circulant families $\mathsf{circ}(\boldsymbol{Z}^1) = \big[\mathsf{circ}(\boldsymbol{z}_1^1), \dots, \mathsf{circ}(\boldsymbol{z}_N^1)\big] \in \mathbb{R}^{d\times(dN)}$? That is, we want to compress and map all these into the same subspace by the ReduNet.

Notice that now the data covariance matrix:

$$\begin{aligned} \mathsf{circ}(\boldsymbol{Z}^1)\mathsf{circ}(\boldsymbol{Z}^1)^\top &= \big[\mathsf{circ}(\boldsymbol{z}_1^1), \dots, \mathsf{circ}(\boldsymbol{z}_N^1)\big]\big[\mathsf{circ}(\boldsymbol{z}_1^1), \dots, \mathsf{circ}(\boldsymbol{z}_N^1)\big]^\top \\ &= \sum_{i=1}^N \mathsf{circ}(\boldsymbol{z}_i^1)\mathsf{circ}(\boldsymbol{z}_i^1)^\top \in \mathbb{R}^{d\times d} \end{aligned} \tag{5.1.22}$$

associated with this family of samples is *automatically* a (symmetric) circulant matrix. Moreover, because the circulant property is preserved under sums, inverses, and products, the matrices $\boldsymbol{E}^1$ and $\boldsymbol{C}_k^1$ are also automatically circulant matrices, whose application to a feature vector $\boldsymbol{z} \in \mathbb{R}^d$ can be implemented using circular convolution "$\circledast$". Specifically, we have the following proposition.

Proposition 5.1 (Convolution structures of $\boldsymbol{E}^1$ and $\boldsymbol{C}_k^1$)**.** *The matrix*

$$\boldsymbol{E}^1 = \alpha\big(\boldsymbol{I} + \alpha\mathsf{circ}(\boldsymbol{Z}^1)\mathsf{circ}(\boldsymbol{Z}^1)^\top\big)^{-1} \tag{5.1.23}$$

is a circulant matrix and represents a circular convolution:

$$\boldsymbol{E}^1\boldsymbol{z} = \boldsymbol{e}_1 \circledast \boldsymbol{z},$$

where $\boldsymbol{e}_1 \in \mathbb{R}^d$ is the first column vector of $\boldsymbol{E}^1$ and "$\circledast$" is circular convolution defined as

$$(\boldsymbol{e}_1 \circledast \boldsymbol{z})_i \doteq \sum_{j=0}^{d-1} e_1(j)x(i+d-j \bmod d).$$

Similarly, the matrices $\boldsymbol{C}_k^1$ associated with any subsets of $\boldsymbol{Z}^1$ are also circular convolutions.

[9] Again, to simplify discussion, we assume for now that the initial features $\boldsymbol{Z}^1$ are $\boldsymbol{X}$ themselves hence have the same dimension d, i.e., $D = d$. But that does not need to be the case as we will soon see that we need to lift $\boldsymbol{X}$ to a higher dimension.

Not only do the first-layer parameters $\boldsymbol{E}^1$ and $\boldsymbol{C}_k^1$ of the ReduNet become circulant convolutions but also the next-layer features remain circulant matrices. That is, the incremental feature transform in (5.1.15) applied to all shifted versions of a $\boldsymbol{z}^1 \in \mathbb{R}^d$, given by

$$\mathsf{circ}(\boldsymbol{z}^1) + \eta \cdot \boldsymbol{E}^1 \mathsf{circ}(\boldsymbol{z}^1) - \eta \cdot \boldsymbol{\sigma}\Big([\boldsymbol{C}_1^1 \mathsf{circ}(\boldsymbol{z}^1), \ldots, \boldsymbol{C}_K^1 \mathsf{circ}(\boldsymbol{z}^1)]\Big), \tag{5.1.24}$$

is a circulant matrix. This implies that there is no need to construct circulant families from the second layer features as we did for the first layer. By denoting

$$\boldsymbol{z}^2 \propto \boldsymbol{z}^1 + \eta \cdot g(\boldsymbol{z}^1, \boldsymbol{\theta}^1) = \boldsymbol{z}^1 + \eta \cdot \boldsymbol{e}_1 \circledast \boldsymbol{z}^1 - \eta \cdot \boldsymbol{\sigma}\Big([\boldsymbol{c}_1^1 \circledast \boldsymbol{z}^1, \ldots, \boldsymbol{c}_K^1 \circledast \boldsymbol{z}^1]\Big), \tag{5.1.25}$$

the features at the next level can be written as

$$\mathsf{circ}(\boldsymbol{Z}^2) = \big[\mathsf{circ}(\boldsymbol{z}_1^1 + \eta g(\boldsymbol{z}_1^1, \boldsymbol{\theta}^1)), \ldots, \mathsf{circ}(\boldsymbol{z}_N^1 + \eta g(\boldsymbol{z}_N^1, \boldsymbol{\theta}^1))\big].$$

Continuing inductively, we see that all matrices $\boldsymbol{E}^\ell$ and $\boldsymbol{C}_k^\ell$ based on such $\mathsf{circ}(\boldsymbol{Z}^\ell)$ are circulant, and so are all features. By virtue of the properties of the data, ReduNet has taken the form of a convolutional network, *with no need to explicitly choose this structure!*

A fundamental trade-off between invariance and sparsity. There is one problem though: In general, the set of all circular permutations of a vector $\boldsymbol{z}$ gives a full-rank matrix. That is, the d "augmented" features associated with each sample (hence each class) typically already span the entire space $\mathbb{R}^d$. For instance, all shifted versions of a delta function $\delta(d)$ can generate any other signal as their (dense) weighted superposition. The MCR^2 objective (4.2.15) will not be able to distinguish classes as different subspaces.

One natural remedy is to improve the separability of the data by "lifting" the original signal to a higher dimensional space, e.g., by taking their responses to multiple, filters $\boldsymbol{k}_1, \ldots, \boldsymbol{k}_C \in \mathbb{R}^d$:

$$\boldsymbol{z}[c] = \boldsymbol{k}_c \circledast \boldsymbol{x} = \mathsf{circ}(\boldsymbol{k}_c)\boldsymbol{x} \in \mathbb{R}^d, \quad c = 1, \ldots, C. \tag{5.1.26}$$

The filters can be pre-designed invariance-promoting filters,[10] or adaptively learned from the data,[11] or randomly selected as we do in our experiments. This operation lifts each original signal $\boldsymbol{x} \in \mathbb{R}^d$ to a C-channel feature, denoted as $\bar{\boldsymbol{z}} \doteq [\boldsymbol{z}[1], \ldots, \boldsymbol{z}[C]]^\top \in \mathbb{R}^{C \times d}$. Then, we may construct the ReduNet on vector representations of $\bar{\boldsymbol{z}}$, denoted as $\vec{(\bar{\boldsymbol{z}})} \doteq [\boldsymbol{z}[1]^\top, \ldots, \boldsymbol{z}[C]^\top] \in \mathbb{R}^{dC}$. The associated circulant version $\mathsf{circ}(\bar{\boldsymbol{z}})$ and its data covariance matrix, denoted as $\bar{\boldsymbol{\Sigma}}(\bar{\boldsymbol{z}})$, for all its shifted versions are given as:

$$\mathsf{circ}(\bar{\boldsymbol{z}}) \doteq \begin{bmatrix} \mathsf{circ}(\boldsymbol{z}[1]) \\ \vdots \\ \mathsf{circ}(\boldsymbol{z}[C]) \end{bmatrix} \in \mathbb{R}^{dC \times d}, \tag{5.1.27}$$

[10] For 1D signals like audio, one may consider the conventional short-time Fourier transform (STFT); for 2D images, one may consider 2D wavelets as in the ScatteringNet [BM13].

[11] For learned filters, one can learn filters as the principal components of samples as in the PCANet [CJG+15] or from convolution dictionary learning [LB19; QLZ19].

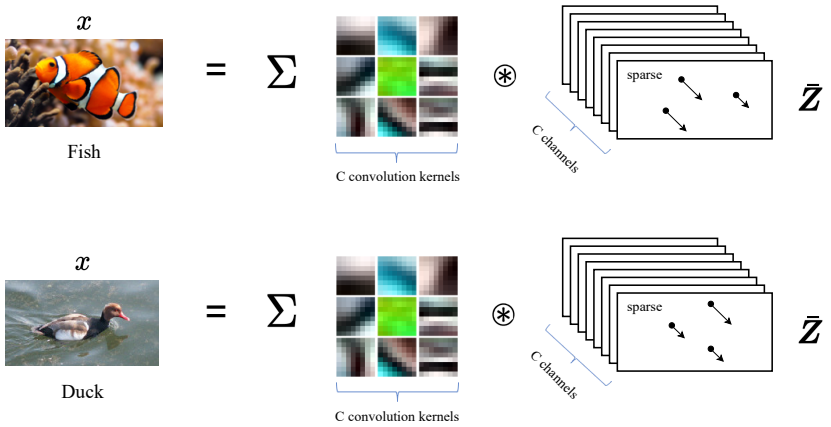


Figure 5.7: Each input signal $\boldsymbol{x}$ (an image here) can be represented as a superposition of sparse convolutions with multiple kernels $\boldsymbol{d}_c$ in a dictionary $\boldsymbol{D}$.

$$\bar{\boldsymbol{\Sigma}}(\bar{\boldsymbol{z}}) \doteq \begin{bmatrix} \mathsf{circ}(\boldsymbol{z}[1]) \\ \vdots \\ \mathsf{circ}(\boldsymbol{z}[C]) \end{bmatrix} \big[\mathsf{circ}(\boldsymbol{z}[1])^\top, \ldots, \mathsf{circ}(\boldsymbol{z}[C])^\top\big] \in \mathbb{R}^{dC \times dC}, \tag{5.1.28}$$

where $\mathsf{circ}(\boldsymbol{z}[c]) \in \mathbb{R}^{d\times d}$ with $c \in [C]$ is the circulant version of the c-th channel of the feature $\bar{\boldsymbol{z}}$. Then the columns of $\mathsf{circ}(\bar{\boldsymbol{z}})$ will only span at most a d-dimensional proper subspace in $\mathbb{R}^{dC}$. However, this simple lifting operation (if linear) is not sufficient to render the classes separable yet—features associated with other classes will span the *same* d-dimensional subspace. This reflects a fundamental conflict between invariance and linear (subspace) modeling: *one cannot hope for arbitrarily shifted and superposed signals to belong to the same class.*

One way of resolving this conflict is to leverage additional structure within each class, in the form of *sparsity*: signals within each class are not generated as an arbitrary linear superposition of some base atoms (or motifs), but only *sparse* combinations of them and their shifted versions, as shown in Figure 5.7. More precisely, let $\boldsymbol{D}_k = [\boldsymbol{d}_{k,1}, \ldots, \boldsymbol{d}_{k,c}]$ denote a matrix with a collection of atoms associated for class k, also known as a dictionary, then each signal $\boldsymbol{x}$ in this class is sparsely generated as:

$$\boldsymbol{x} = \boldsymbol{d}_{k,1} \circledast z_1 + \ldots + \boldsymbol{d}_{k,c} \circledast z_c = \mathsf{circ}(\boldsymbol{D}_k)\boldsymbol{z}, \tag{5.1.29}$$

for some sparse vector $\boldsymbol{z}$. Signals in different classes are then generated by different dictionaries whose atoms (or motifs) are incoherent from one another. Due to incoherence, signals in one class are unlikely to be sparsely represented by atoms in any other class. Hence all signals can be represented as

$$\boldsymbol{x} = \big[\mathsf{circ}(\boldsymbol{D}_1), \mathsf{circ}(\boldsymbol{D}_2), \ldots, \mathsf{circ}(\boldsymbol{D}_K)\big]\bar{\boldsymbol{z}}, \tag{5.1.30}$$

where $\bar{\boldsymbol{z}}$ is sparse.[12] There is a vast literature on how to learn the most compact and optimal sparsifying dictionaries from sample data, e.g. [LB19; QLZ19] and subsequently solve the inverse problem and compute the associated sparse code $\boldsymbol{z}$ or $\bar{\boldsymbol{z}}$. Recent studies of [QLZ20; QZL+20a] even show that under broad conditions the convolution dictionary learning problem can be solved effectively and efficiently.

Nevertheless, for tasks such as classification, we are not necessarily interested in the precise optimal dictionary nor the precise sparse code for each individual signal. We are mainly interested if collectively the set of sparse codes for each class are adequately separable from those of other classes. Under the assumption of the sparse generative model, if the convolution kernels $\{\boldsymbol{k}_c\}_{c=1}^C$ match well with the "transpose" or "inverse" of the above sparsifying dictionaries $\boldsymbol{D} = [\boldsymbol{D}_1, \ldots, \boldsymbol{D}_K]$, also known as the *analysis filters* [NDE+13; RE14], signals in one class will only have high responses to a small subset of those filters and low responses to others (due to the incoherence assumption). Nevertheless, in practice, often a sufficiently large number of, say C, random filters $\{\boldsymbol{k}_c\}_{c=1}^C$ suffices to ensure that the extracted C-channel features

$$\big[\boldsymbol{k}_1 \circledast \boldsymbol{x}, \boldsymbol{k}_2 \circledast \boldsymbol{x}, \ldots, \boldsymbol{k}_C \circledast \boldsymbol{x}\big]^\top = \big[\mathsf{circ}(\boldsymbol{k}_1)\boldsymbol{x}, \ldots, \mathsf{circ}(\boldsymbol{k}_C)\boldsymbol{x}\big]^\top \in \mathbb{R}^{C\times d} \tag{5.1.31}$$

for different classes have different response patterns to different filters hence make different classes separable [CJG+15].

Therefore, in our framework, to a large extent the number of channels (or the width of the network) truly plays the role as the *statistical resource* whereas the number of layers (the depth of the network) plays the role as the *computational resource*. The theory of compressive sensing precisely characterizes how many measurements are needed in order to preserve the intrinsic low-dimensional structures (including separability) of the data [WM22].

The multi-channel responses $\bar{\boldsymbol{z}}$ should be sparse. So to approximate the sparse code $\bar{\boldsymbol{z}}$, we may take an entry-wise *sparsity-promoting nonlinear thresholding*, say $\boldsymbol{\tau}(\cdot)$, on the above filter outputs by setting low (say absolute value below ϵ) or negative responses to be zero:

$$\bar{\boldsymbol{z}} \doteq \boldsymbol{\tau}\Big(\big[\mathsf{circ}(\boldsymbol{k}_1)\boldsymbol{x}, \ldots, \mathsf{circ}(\boldsymbol{k}_C)\boldsymbol{x}\big]^\top\Big) \in \mathbb{R}^{C\times d}. \tag{5.1.32}$$

Figure 5.8 illustrates the basic ideas. One may refer to [RE14] for a more systematical study on the design of the sparsifying thresholding operator. Nevertheless, here we are not so interested in obtaining the best sparse codes as long as the codes are sufficiently separable. Hence the nonlinear operator $\boldsymbol{\tau}$ can be simply chosen to be a soft thresholding or a ReLU. These presumably sparse features $\bar{\boldsymbol{z}}$ can be assumed to lie on a lower-dimensional (nonlinear) submanifold of $\mathbb{R}^{dC}$, which can be linearized and separated from the other classes by subsequent ReduNet layers, as illustrated later in Figure 5.9.

[12] Notice that similar sparse representation models have long been proposed and used for classification purposes in applications such a face recognition, demonstrating excellent effectiveness [WWG+12; WYG+09]. Recently, the convolution sparse coding model has been proposed by [PRE17] as a framework for interpreting the structures of deep convolution networks.

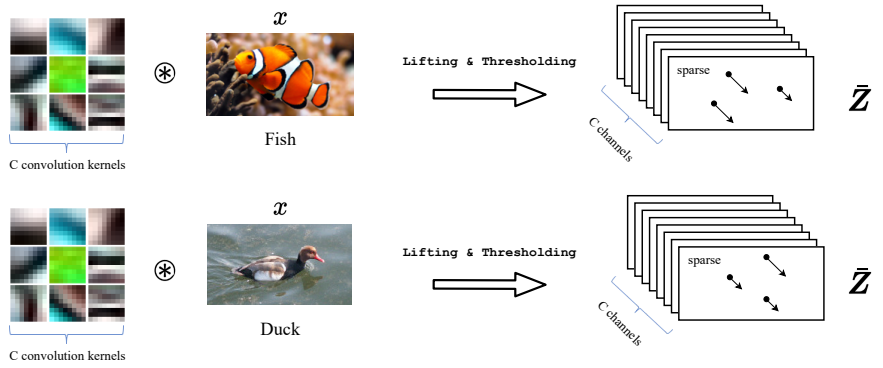


Figure 5.8: Estimate the sparse code $\bar{\boldsymbol{z}}$ of an input signal $\boldsymbol{x}$ (an image here) by taking convolutions with multiple kernels $\boldsymbol{k}_c$ and then sparsifying.

The ReduNet constructed from circulant version of these multi-channel features $\bar{\boldsymbol{Z}} \doteq [\bar{\boldsymbol{z}}_1, \ldots, \bar{\boldsymbol{z}}_N] \in \mathbb{R}^{C \times d \times N}$, i.e., $\mathsf{circ}(\bar{\boldsymbol{Z}}) \doteq [\mathsf{circ}(\bar{\boldsymbol{z}}_1), \ldots, \mathsf{circ}(\bar{\boldsymbol{z}}_N)] \in \mathbb{R}^{dC \times dN}$, retains the good invariance properties described above: the linear operators, now denoted as $\bar{\boldsymbol{E}}$ and $\bar{\boldsymbol{C}}_k$, remain block circulant, and represent *multi-channel 1D circular convolutions.* Specifically, we have the following result.

Proposition 5.2 (Multi-channel convolution structures of $\bar{\boldsymbol{E}}$ and $\bar{\boldsymbol{C}}_k$)**.** *The matrix*

$$\bar{\boldsymbol{E}} \doteq \alpha \left(\boldsymbol{I} + \alpha\, \mathsf{circ}(\bar{\boldsymbol{Z}}) \mathsf{circ}(\bar{\boldsymbol{Z}})^\top \right)^{-1} \tag{5.1.33}$$

is block circulant, i.e.,

$$\bar{\boldsymbol{E}} = \begin{bmatrix} \bar{\boldsymbol{E}}_{1,1} & \cdots & \bar{\boldsymbol{E}}_{1,C} \\ \vdots & \ddots & \vdots \\ \bar{\boldsymbol{E}}_{C,1} & \cdots & \bar{\boldsymbol{E}}_{C,C} \end{bmatrix} \in \mathbb{R}^{dC \times dC}, \tag{5.1.34}$$

where each $\bar{\boldsymbol{E}}_{c,c'} \in \mathbb{R}^{d \times d}$ is a circulant matrix. Moreover, $\bar{\boldsymbol{E}}$ represents a multi-channel circular convolution, i.e., for any multi-channel signal $\bar{\boldsymbol{z}} \in \mathbb{R}^{C \times n}$ we have

$$\bar{\boldsymbol{E}} \cdot \mathsf{vec}(\bar{\boldsymbol{z}}) = \mathsf{vec}(\bar{\boldsymbol{e}} \circledast \bar{\boldsymbol{z}}). \tag{5.1.35}$$

In above, $\bar{\boldsymbol{e}} \in \mathbb{R}^{C \times C \times d}$ is a multi-channel convolutional kernel with $\bar{\boldsymbol{e}}[c, c'] \in \mathbb{R}^d$ being the first column vector of $\bar{\boldsymbol{E}}_{c,c'}$, and $\bar{\boldsymbol{e}} \circledast \bar{\boldsymbol{z}} \in \mathbb{R}^{C \times d}$ is the multi-channel circular convolution defined as

$$(\bar{\boldsymbol{e}} \circledast \bar{\boldsymbol{z}})[c] \doteq \sum_{c'=1}^{C} \bar{\boldsymbol{e}}[c, c'] \circledast \bar{\boldsymbol{z}}[c'], \quad \forall c = 1, \ldots, C. \tag{5.1.36}$$

Similarly, the matrices $\bar{\boldsymbol{C}}_k$ associated with any subsets of $\bar{\boldsymbol{Z}}$ are also multi-channel circular convolutions.

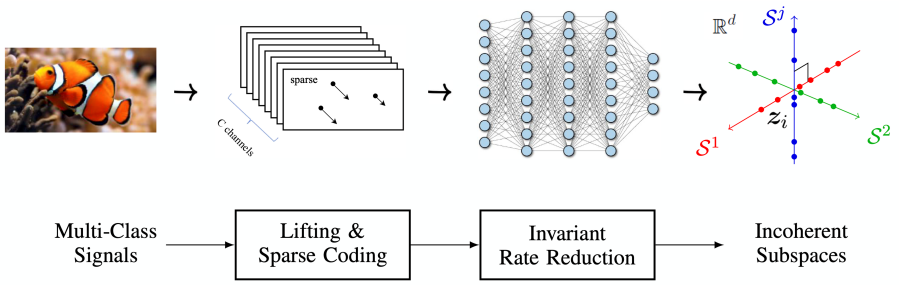


Figure 5.9: The overall process for classifying multi-class signals with shift invariance: Multi-channel lifting, sparse coding, followed by a multi-channel convolution ReduNet for invariant rate reduction. These components are *necessary* in order to map shift-invariant multi-class signals to incoherent (linear) subspaces as an LDR. Note that the architectures of most modern deep neural networks resemble this process. The so-learned LDR facilitates subsequent tasks such as classification.

From Proposition 5.2, shift invariant ReduNet is a deep convolutional network for multi-channel 1D signals by construction. Notice that even if the initial lifting kernels are separated (5.1.32), the matrix inverse in (5.1.33) for computing $\bar{\boldsymbol{E}}$ (similarly for $\bar{\boldsymbol{C}}_k$) introduces "cross talk" among all C channels. Hence, these multi-channel convolutions in general are *not* depth-wise separable, unlike the Xception nets [Cho17] that were once suggested to simplify multi-channel convolutional neural networks.[13]

Remark 5.2 (Reducing Computational Complexity in the Frequency Domain). The calculation of $\bar{\boldsymbol{E}}$ in (5.1.33) requires inverting a matrix of size $dC \times dC$, which in general has complexity $O(d^3C^3)$. Nevertheless, by using the fact that a circulant matrix can be diagonalized by the Discrete Fourier Transform (DFT) matrix, the complexity can be significantly reduced. As shown in [CYY+22], to compute $\bar{\boldsymbol{E}}$ and $\bar{\boldsymbol{C}}_k \in \mathbb{R}^{dC \times dC}$, we only need to compute in the frequency domain the inverse of $C \times C$ blocks for d times hence the overall complexity becomes $O(dC^3)$.

Overall network architecture and comparison. Following the above derivation, we see that in order to find a linear discriminative representation (LDR) for multiple classes of signals/images that is invariant to translation, sparse coding, a multi-layer architecture with multi-channel convolutions, different nonlinear activation, and spectrum computing all become *necessary* components for achieving the objective effectively and efficiently. Figure 5.9 illustrates the overall process of learning such a representation via invariant rate reduction on the input sparse codes.

[13] It remains open what additional structures on the data would lead to depth-wise separable convolutions.

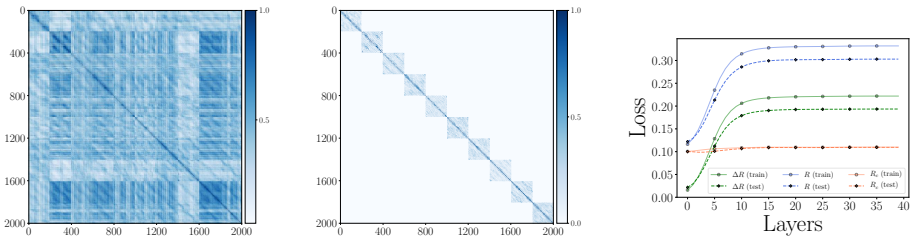
(a) $\boldsymbol{X}_{\text{rotation}}$ (b) $\boldsymbol{Z}_{\text{rotation}}$ (c) Loss

Figure 5.10: Examples of rotated images of MNIST digits, each by 18°. (**Left**) Diagram for polar coordinate representation; (**Right**) Rotated images of digit '0' and '1'.

Example 5.2 (Invariant Classification of Digits). We next provide an empirical performance of the ReduNet on learning *rotation* invariant features on the real 10-class MNIST dataset. We impose a polar grid on the image $\boldsymbol{x} \in \mathbb{R}^{H \times W}$, with its geometric center being the center of the 2D polar grid (as illustrated in Figure 5.10). For each radius r_i, $i \in [C]$, we can sample Γ pixels with respect to each angle $\gamma_l = l \cdot (2\pi/\Gamma)$ with $l \in [\Gamma]$. Then given a sample image $\boldsymbol{x}$ from the dataset, we represent the image in the (sampled) polar coordinate as a multi-channel signal $\boldsymbol{x}_p \in \mathbb{R}^{\Gamma \times C}$. The goal here is to learn a rotation invariant representation, i.e., we expect to learn $f(\cdot, \theta)$ such that $\{f(\boldsymbol{x}_p \circ \mathfrak{g}, \theta)\}_{\mathfrak{g} \in \mathbb{G}}$ lie in the same subspace, where $\mathfrak{g}$ is the cyclic-shift in polar angle. We use $N = 100$ training samples (10 from each class) and set $\Gamma = 200$, $C = 15$ for polar sampling. By performing the above sampling in polar coordinate, we can obtain the data matrix $\boldsymbol{X}_p \in \mathbb{R}^{(\Gamma \cdot C) \times N}$. For the ReduNet, we set the number of layers/iterations $L = 40$, precision $\epsilon = 0.1$, step size $\eta = 0.5$. Before the first layer, we perform lifting of the input by 1D circulant-convolution with 20 random Gaussian kernels of size 5.

To evaluate the learned representation, each training sample is augmented by 20 of its rotated version, each shifted with stride=10. We compute the cosine similarities among the $m \times 20$ augmented training inputs $\boldsymbol{X}_{\text{rotation}}$ and the results are shown in Figure 5.11 (**a**). We compare the cosine similarities among the learned features of all the augmented versions, i.e., $\bar{\boldsymbol{Z}}_{\text{rotation}}$ and summarize the results in Figure 5.11 (**b**). As we see, the so-constructed rotation-invariant ReduNet is able to map the training data (as well as all its rotated versions) from the 10 different classes into 10 nearly orthogonal subspaces. That is, the learned subspaces are truly invariant to shift transformation in polar angle. Next, we randomly draw another 100 test samples followed by the same augmentation procedure. In Figure 5.11 (**c**), we visualize the MCR^2 loss on the ℓ-th layer representation of the ReduNet on the training and test dataset. From these results, we can find that the constructed ReduNet is indeed able to maximize the MCR^2 loss as well as generalize to the test data.

■



(a) $\boldsymbol{X}_{\text{rotation}}$ (b) $\boldsymbol{Z}_{\text{rotation}}$ (c) Loss

Figure 5.11: (a)(b) are heatmaps of cosine similarity among rotated training data $\boldsymbol{X}_{\text{rotation}}$ and learned features $\bar{\boldsymbol{Z}}_{\text{rotation}}$ for rotation invariance. (d) visualizes the training/val MCR^2 losses across layers.

5.2 White-Box Transformers from Unrolled Optimization

As we have seen in the previous section, we use the problem of classification to provide a rigorous interpretation for main architectural characteristics of popular deep networks such as the ResNet and the CNN: each layer of such networks can be viewed as to imitate a gradient step which increases the rate reduction (or information gain) objective. This perspective also leads to a somewhat surprising fact: the the parameters and operators of the layers of such a deep network, the ReduNet, can be computed in a purely forward fashion.

Despite the theoretical and conceptual importance of the ReduNet, several factors limit it from being very practical. First, as we have discussed in the above, the computational cost of computing the matrix operators in each layer in a forward fashion can be very high. Second, the so-computed operators may not be so effective in optimizing the objective and it might take thousands of iterations (hence layers). As we have seen in Section 2.3.3 for LISTA, these two issues can be addressed by allowing to optimize those operators and make them learnable via back-propagation.[14]

The supervised classification setting in which the ReduNet was derived is also somewhat limiting. In practice, an image might not belong to a single class as it may contain multiple objects. Hence it would be more general to assume that different regions of the image belong to different low-dimensional models (say a Gaussian or a subspace). As we will see, such a generalization would lead to a both simple and general architecture which unifies the rate reduction and the denoising operations that we have seen in the previous chapter. Moreover, the so-obtained architecture resembles the popular Transformer architecture.

[14] Or, perhaps, by a mixture of both forward and backward optimization.

5.2.1 Unrolled Optimization for Sparse Rate Reduction

For the past several years, as the amount and quality of data available for training deep networks has increased dramatically, the field has moved from considering inputs as "atoms" to inputs as "sequences of tokens", where each token is a part of the overall input. This shift in perspective has both philosophical and pragmatic improvements: philosophically, it allows us to measure and model each part of the input and their interactions; pragmatically, it allows us to use similar deep networks, *sequence-to-sequence models* such as the ubiquitous Transformers, to learn every kind of data distribution. Now, it falls to us to understand representation learning in the "token-centric" model of data as token sequences, and to cope with the aforementioned challenges (such as, in the context of the rate reduction framework, not having labels for each token).

Formally, let $\boldsymbol{X} = \big[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\big] \in \mathbb{R}^{D \times N}$ denote random variables representing our data source. In vision tasks, each $\boldsymbol{x}_i \in \mathbb{R}^D$ is interpreted as a *token*, typically corresponding to an image patch. In language tasks, each $\boldsymbol{x}_i \in \mathbb{R}^D$ is interpreted as an *token embedding*, i.e., a continuous vector representation of a discrete token such as a word or subword.[15] The $\boldsymbol{x}_i$'s may have arbitrary correlation structures. We use $\boldsymbol{Z} = \big[\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\big] \in \mathbb{R}^{d \times N}$ to denote the random variables that defines our representations, where $\boldsymbol{z}_i \in \mathbb{R}^d$ is the representation of the corresponding token $\boldsymbol{x}_i \in \mathbb{R}^D$.

Remark 5.3. In transformers, each input sample is typically converted into a sequence of *tokens*. A token is a basic unit of information derived from the raw input: in natural language processing, tokens are typically words or subwords; in computer vision, they correspond to image patches; and in other modalities, they may represent time steps, spatial locations, or other domain-specific units. A *token embedding* is a continuous vector representation of a token that serves as the input to a transformer. It maps each token to a point in a high-dimensional space, enabling the model to process symbolic inputs using numerical computation. A *token representation* is a vector that encodes the semantic or structural information of a token, typically produced by the intermediate or final layers of a transformer. These representations are designed to capture meaningful features of the input that are useful for downstream tasks such as classification, generation, or regression. Please refer to Section 8.2 for more details about this vocabulary and how it relates to implementations.

Objective for learning a structured and compact representation. Inspired by the previous discussion of rate reduction (Section 5.1), we propose that the objective of representation learning in this token-centric model for our data is to find a feature mapping $f\colon \boldsymbol{X} \in \mathbb{R}^{D \times N} \to \boldsymbol{Z} \in \mathbb{R}^{d \times N}$ which transforms input tokens $\{\boldsymbol{x}_i\}_{i=1}^N \subset \mathbb{R}^D$ with a potentially nonlinear and multi-modal distribution to a (piecewise) *linearized and compact* token representations $\{\boldsymbol{z}_i\}_{i=1}^N \subset \mathbb{R}^d$. While the *joint* distribution of tokens representations $\{\boldsymbol{z}_i\}_{i=1}^N$ may be sophisticated (and task-specific), we further contend that it is reasonable and practical

[15] With a slight abuse of terminology, we refer to both the discrete tokens and their associated embeddings simply as tokens throughout this chapter for convenience.

to require that the target *marginal* distribution of individual token representations should be highly compressed and structured, amenable for compact coding. Particularly, we require the distribution to be *a mixture of low-dimensional (say K) Gaussian distributions*, such that the k-th Gaussian has mean $\mathbf{0} \in \mathbb{R}^d$, covariance $\boldsymbol{\Sigma}_k \succeq \mathbf{0} \in \mathbb{R}^{d\times d}$, and support spanned by the orthonormal basis $\boldsymbol{U}_k \in \mathbb{R}^{d\times p}$. We denote $\boldsymbol{U}_{[K]} = \{\boldsymbol{U}_k\}_{k=1}^K$ to be the set of bases of all Gaussians.

As usual, in order to learn good representations, we wish to maximize the *information gain* for the final token representations, or more specifically to maximize an appropriate notion of the rate reduction (see Section 4.2.3), i.e.,

$$\max_{\boldsymbol{Z}\in\mathbb{R}^{d\times N}} \Delta R_\epsilon(\boldsymbol{Z} \mid \boldsymbol{U}_{[K]}) \doteq R_\epsilon(\boldsymbol{Z}) - R_\epsilon^c(\boldsymbol{Z} \mid \boldsymbol{U}_{[K]}). \tag{5.2.1}$$

Here, the first term R_ϵ is an estimate of the lossy coding rate for the whole set of token representations. More specifically, if we view the token representations $\{\boldsymbol{z}_i\}_{i=1}^N$ as i.i.d. samples from a single zero-mean Gaussian, their lossy coding rate subject to a quantization precision $\epsilon > 0$ is given as

$$R_\epsilon(\boldsymbol{Z}) \doteq \frac{1}{2}\text{logdet}\left(\boldsymbol{I} + \frac{d}{N\epsilon^2}\boldsymbol{Z}^\top\boldsymbol{Z}\right) = \frac{1}{2}\text{logdet}\left(\boldsymbol{I} + \frac{d}{N\epsilon^2}\boldsymbol{Z}\boldsymbol{Z}^\top\right). \tag{5.2.2}$$

The second term R_ϵ^c is an estimate of the lossy coding rate under the codebook $\boldsymbol{U}_{[K]}$, which is given as

$$R_\epsilon^c(\boldsymbol{Z} \mid \boldsymbol{U}_{[K]}) \doteq \sum_{k=1}^K R_\epsilon(\boldsymbol{U}_k^\top\boldsymbol{Z}) = \frac{1}{2}\sum_{k=1}^K \log\det\left(\boldsymbol{I} + \frac{p}{N\epsilon^2}(\boldsymbol{U}_k^\top\boldsymbol{Z})^\top(\boldsymbol{U}_k^\top\boldsymbol{Z})\right). \tag{5.2.3}$$

Remark 5.4. The expression (5.2.3) for the coding rate can be viewed as a generalization of the coding rate R_ϵ^c used in the original rate reduction objective (4.2.16). In particular, the original objective is defined with respect to a set of known membership labels $\{\boldsymbol{\Pi}_k\}$ specific to the particular data realization $\boldsymbol{X}$. In contrast, the current objective is defined with respect to subspaces $\boldsymbol{U}_{[K]}$, which are independent of any particular realization but are assumed to support the distribution of token representations. Suppose that a token representation $\boldsymbol{z}_i$ belongs to a subspace $\boldsymbol{U}_k$ and these subspaces are approximately orthogonal to each other, i.e., $\boldsymbol{U}_k^\top\boldsymbol{U}_l \approx \mathbf{0}$ for all $k \neq l$. Then, one can verify that the projections $\boldsymbol{U}_k\boldsymbol{U}_k^\top\boldsymbol{z}_i = \boldsymbol{z}_i$ and $\boldsymbol{U}_l\boldsymbol{U}_l^\top\boldsymbol{z}_i \approx \mathbf{0}$ for all $l \neq k$. These orthogonal projections effectively serve as implicit membership labels, identifying the subspace to which each token representation belongs.

Remark 5.5. Notably, this set of desiderata for representation learning aligns well with two well-established empirical hypotheses about the structure of token representations in trained models which use the ubiquitous transformer neural network architecture: the "linear representation hypothesis" [JRR+24; PCV24] and the "superposition hypothesis" [EHO+22a; YCO+21]. The linear representation hypothesis posits that token representations in transformer models lie in low-dimensional linear subspaces that encode semantic features. Similarly, the superposition hypothesis suggests that these representations can be approximately expressed as a sparse linear combination of these feature vectors. As we

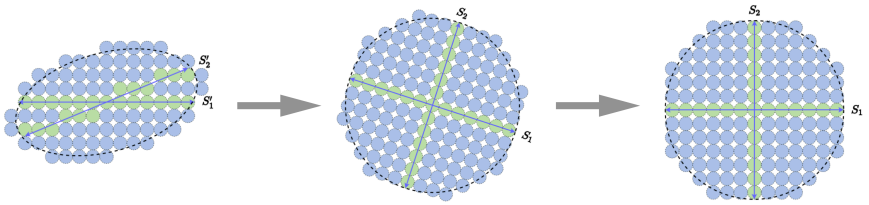


Figure 5.12: **Comparison of three sets of representations via rate reduction and sparsity.** Each S_i represents one linear subspace, and the number of blue balls represents the difference between the coding rates $\Delta R_\epsilon(\boldsymbol{Z} \mid \boldsymbol{U}_{[K]}) = R_\epsilon(\boldsymbol{Z}) - R_\epsilon^c(\boldsymbol{Z} \mid \boldsymbol{U}_{[K]})$.

will see in Figure 5.17 later in this Chapter, the subspace basis $\boldsymbol{U}_k$ may be interpreted as a set of semantic features, where each feature corresponds to a specific aspect of the token's meaning. Token representations are then approximately expressed as sparse linear combinations of these subspace bases, capturing the essential semantic components of the token while ignoring irrelevant dimensions.

Sparse rate reduction. Note that the rate reduction objective (5.2.1) is invariant to arbitrary joint rotations of the representations and subspaces. In particular, optimizing the rate reduction objective may not naturally lead to axis-aligned (i.e., *sparse*) representations. For instance, consider the three sets of learned representations in Figure 5.12. The coding rate reduction increases from (a) to (b), but because it is invariant under rotations, remains the same from (b) to (c). Therefore, we would like to transform the representations (and their supporting subspaces) so that the representations $\boldsymbol{Z}$ eventually become sparse[16] with respect to the standard coordinates of the resulting representation space as in Figure 5.12(c). Computationally, we may combine the above two goals into a unified objective for optimization:

$$\max_{f \in \mathcal{F}} \left[\Delta R_\epsilon(\boldsymbol{Z} \mid \boldsymbol{U}_{[K]}) - \lambda \|\boldsymbol{Z}\|_0\right] \qquad \text{s.t. } \boldsymbol{Z} = f(\boldsymbol{X}), \tag{5.2.4}$$

where $\mathcal{F}$ denotes a general function class and the ℓ_0 norm $\|\boldsymbol{Z}\|_0$ promotes the sparsity of the final token representations $\boldsymbol{Z} = f(\boldsymbol{X})$.

In practice, the ℓ_0 norm is often relaxed to the ℓ_1 norm to improve computational tractability and enable convex optimization techniques [WM22]. Motivated by this, we relax Problem (5.2.4) accordingly, leading to a formulation that remains faithful to the original sparsity objective while being more amenable to efficient algorithms as follow:

$$\max_{f \in \mathcal{F}} \left[\Delta R_\epsilon(\boldsymbol{Z} \mid \boldsymbol{U}_{[K]}) - \lambda \|\boldsymbol{Z}\|_1\right] \qquad \text{s.t. } \boldsymbol{Z} = f(\boldsymbol{X}), \tag{5.2.5}$$

With a slight abuse of terminology, we often refer to this objective function also as the *sparse rate reduction*.

[16] Concretely, having few nonzero entries.

White-box network architecture via unrolled optimization. Although easy to state, each term in the above objective is computationally challenging to optimize [WM22]. Hence it is natural to adopt an approximation approach that realizes the global transformation f to optimize (5.2.4) through a concatenation of multiple, say L, simple *incremental and local* operations f^ℓ that push the representation distribution towards the desired parsimonious model distribution:

$$f\colon \boldsymbol{X} \xrightarrow{f^{\text{pre}}} \boldsymbol{Z}^1 \to \cdots \to \boldsymbol{Z}^\ell \xrightarrow{f^\ell} \boldsymbol{Z}^{\ell+1} \to \cdots \xrightarrow{f^L} \boldsymbol{Z}^{L+1} = \boldsymbol{Z}, \tag{5.2.6}$$

where $f^{\text{pre}} : \mathbb{R}^D \to \mathbb{R}^d$ is the pre-processing mapping that transforms each input token $\boldsymbol{x}_i \in \mathbb{R}^D$ to the initial token representations $\boldsymbol{z}_i^1 \in \mathbb{R}^d$. Each incremental *forward mapping* $\boldsymbol{Z}^{\ell+1} = f^\ell(\boldsymbol{Z}^\ell)$, or a "layer", transforms the token distribution to *optimize* the above sparse rate reduction objective (5.2.4), conditioned on the distribution of its input $\boldsymbol{Z}^\ell$.

Remark 5.6. In contrast to other unrolled optimization approaches such as the ReduNet (see Section 5.1), we *explicitly model* the distribution of $\boldsymbol{Z}^\ell$ at each layer, say as a mixture of linear subspaces or sparsely generated from a dictionary. The model parameters are learned from data (say via *backward propagation* with end-to-end training). This separation between forward "optimization" and backward "learning" clarifies the mathematical role of each layer as an operator that transforms the distribution of its input, whereas the input distribution is in turn modeled (and subsequently learned) by the parameters of the layer.

Now, we show how to derive these incremental and local operations through an unrolled optimization perspective to solve Problem (5.2.5). Once we decide on using an incremental approach to optimizing Problem (5.2.5), there are a variety of possible choices to achieve the optimization. Given a model for $\boldsymbol{Z}^\ell$, say a mixture of subspaces $\boldsymbol{U}_{[K]}$, we opt for a two-step *alternating minimization* method with a strong conceptual basis. First, we *compress* the tokens $\boldsymbol{Z}^\ell$ via a gradient descent to minimize the coding rate term $R_\epsilon^c(\boldsymbol{Z} \mid \boldsymbol{U}_{[K]}^\ell)$. Specifically, we take a gradient step on R_ϵ^c with a learning rate κ as follows:

$$\boldsymbol{Z}^{\ell+1/2} = \boldsymbol{Z}^\ell - \kappa \nabla_{\boldsymbol{Z}} R_\epsilon^c(\boldsymbol{Z} \mid \boldsymbol{U}_{[K]}^\ell). \tag{5.2.7}$$

Next, we *sparsify* the compressed tokens, generating $\boldsymbol{Z}^{\ell+1}$ via a suitably-relaxed proximal gradient step to minimize the remaining term $\lambda\|\boldsymbol{Z}\|_1 - R_\epsilon(\boldsymbol{Z})$. As we will argue in detail later, we can find such a $\boldsymbol{Z}^{\ell+1}$ by solving a sparse presentation problem with respect to an orthogonal complete dictionary $\boldsymbol{D}^\ell$:

$$\boldsymbol{Z}^{\ell+1} = \arg\min_{\boldsymbol{Z}} \left\{ \lambda\|\boldsymbol{Z}\|_1 + \frac{1}{2}\|\boldsymbol{Z}^{\ell+1/2} - \boldsymbol{D}^\ell \boldsymbol{Z}\|_F^2 \right\}. \tag{5.2.8}$$

In the following, we provide technical details for each of the two steps above and derive efficient updates for their implementation.

Self-attention as gradient descent on coding rate of token representations. For the first step (5.2.7), the gradient of the coding rate $\nabla_{\boldsymbol{Z}} R_\epsilon^c$ is costly to compute, as it involves K separate matrix inverses, one for each of the K subspaces with basis $\boldsymbol{U}_k^\ell$:

$$\nabla_{\boldsymbol{Z}} R_\epsilon^c(\boldsymbol{Z} \mid \boldsymbol{U}_{[K]}) = \frac{p}{N\epsilon^2} \sum_{k=1}^{K} \boldsymbol{U}_k \boldsymbol{U}_k^\top \boldsymbol{Z} \Big(\boldsymbol{I} + \frac{p}{N\epsilon^2} (\boldsymbol{U}_k^\top \boldsymbol{Z})^\top (\boldsymbol{U}_k^\top \boldsymbol{Z}) \Big)^{-1}. \tag{5.2.9}$$

Now, we demonstrate that this gradient can be naturally approximated using a so-called multi-head subspace self-attention (MSSA) operator, which has a similar functional form to the multi-head self-attention operator [VSP+17b] with K heads (i.e., one for each subspace, coming from each matrix inverse). Here, we approximate the gradient (5.2.9) using the first-order Neumann series (see Exercise 5.2):[17]

$$\nabla_{\boldsymbol{Z}} R_\epsilon^c(\boldsymbol{Z} \mid \boldsymbol{U}_{[K]}) \tag{5.2.10}$$

$$\approx \frac{p}{N\epsilon^2} \sum_{k=1}^{K} \boldsymbol{U}_k \boldsymbol{U}_k^\top \boldsymbol{Z} \Big(\boldsymbol{I} - \frac{p}{N\epsilon^2} (\boldsymbol{U}_k^\top \boldsymbol{Z})^\top (\boldsymbol{U}_k^\top \boldsymbol{Z}) \Big) \tag{5.2.11}$$

$$= \frac{p}{N\epsilon^2} \Big(\sum_{k=1}^{K} \boldsymbol{U}_k \boldsymbol{U}_k^\top \Big) \boldsymbol{Z} - \Big(\frac{p}{N\epsilon^2} \Big)^2 \sum_{k=1}^{K} \boldsymbol{U}_k (\boldsymbol{U}_k^\top \boldsymbol{Z}) (\boldsymbol{U}_k^\top \boldsymbol{Z})^\top (\boldsymbol{U}_k^\top \boldsymbol{Z}). \tag{5.2.12}$$

In the above approximation, we compute the similarity between projected token representations $\{\boldsymbol{U}_k^\top \boldsymbol{z}_i\}_{i=1}^n$ through an auto-correlation among the projected features as $(\boldsymbol{U}_k^\top \boldsymbol{Z})^\top (\boldsymbol{U}_k^\top \boldsymbol{Z})$. Even if each token feature within $\boldsymbol{Z}$ is normalized, the norm of this matrix can grow arbitrarily[18] as the sequence length increases, making our Neumann approximation increasingly poor (as the true gradient does not grow arbitrarily in size with sequence length). To fix this practical issue and generalize our formulation, we can instead use an arbitrary kernel to measure similarity. One particularly fitting kernel that aligns with our motivation of only auto-regressing against tokens which belong to the same subspace is the local (Nadaraya-Watson) kernel which involves membership estimation via softmax [CMP+21], namely computing (relative) similarity as $\mathrm{softmax}((\boldsymbol{U}_k^\top \boldsymbol{Z})^\top (\boldsymbol{U}_k^\top \boldsymbol{Z}))$. Now suppose that the subspaces expand "as much as possible", that is, all subspaces' bases $\boldsymbol{U}_{[K]}$ together span the whole space. Then, we have $\sum_{k=1}^K \boldsymbol{U}_k \boldsymbol{U}_k^\top = \boldsymbol{I}$. Hence, (5.2.10) becomes

$$\nabla_{\boldsymbol{Z}} R_\epsilon^c(\boldsymbol{Z} \mid \boldsymbol{U}_{[K]}) \approx \frac{p}{N\epsilon^2} \boldsymbol{Z} - \Big(\frac{p}{N\epsilon^2} \Big)^2 \mathrm{MSSA}(\boldsymbol{Z}^\ell \mid \boldsymbol{U}_{[K]}^\ell), \tag{5.2.13}$$

[17] This approximation is only meaningful when $\boldsymbol{Z}$ has a small norm, which in practice can be enforced by normalization strategies (such as the so-called "layer normalization"). We do not include it here for notational convenience, though the corresponding layer normalization operator is included in the final architecture.

[18] Given the practical choice of $\epsilon^2 = p/N$, i.e., a more precise code is used to accommodate more information from more samples.

where MSSA is defined through an SSA operator as follows:

$$\mathrm{SSA}(\boldsymbol{Z} \mid \boldsymbol{U}_k) \doteq (\boldsymbol{U}_k^\top \boldsymbol{Z})\,\mathrm{softmax}((\boldsymbol{U}_k^\top \boldsymbol{Z})^\top (\boldsymbol{U}_k^\top \boldsymbol{Z})),\ \forall k \in [K], \tag{5.2.14}$$

$$\mathrm{MSSA}(\boldsymbol{Z} \mid \boldsymbol{U}_{[K]}) \doteq \frac{p}{N\epsilon^2}\,[\boldsymbol{U}_1, \ldots, \boldsymbol{U}_K] \begin{bmatrix} \mathrm{SSA}(\boldsymbol{Z} \mid \boldsymbol{U}_1) \\ \vdots \\ \mathrm{SSA}(\boldsymbol{Z} \mid \boldsymbol{U}_K) \end{bmatrix}. \tag{5.2.15}$$

Substituting (5.2.13) into (5.2.7) yields that it can naturally be approximated by

$$\boldsymbol{Z}^{\ell+1/2} = \left(1 - \frac{\kappa p}{N\epsilon^2}\right)\boldsymbol{Z}^\ell + \frac{\kappa p}{N\epsilon^2}\mathrm{MSSA}(\boldsymbol{Z}^\ell \mid \boldsymbol{U}_{[K]}^\ell). \tag{5.2.16}$$

Remark 5.7. The SSA operator in (5.2.14) resembles the *attention operator* in a typical transformer [VSP+17b], except that here the linear operators of value, key, and query are all set to be *the same* as the subspace basis, i.e., $\boldsymbol{V}_k = \boldsymbol{K}_k = \boldsymbol{Q}_k = \boldsymbol{U}_k^*$. Hence, we name $\mathrm{SSA}(\cdot \mid \boldsymbol{U}_k) : \mathbb{R}^{d\times n} \to \mathbb{R}^{p\times n}$ the **S**ubspace **S**elf-**A**ttention (SSA) operator. Then, the whole MSSA operator in (5.2.15), formally defined as $\mathrm{MSSA}(\cdot \mid \boldsymbol{U}_{[K]}) \colon \mathbb{R}^{d\times n} \to \mathbb{R}^{d\times n}$ and called the **M**ulti-Head **S**ubspace **S**elf-**A**ttention (MSSA) operator, aggregates the attention head outputs by averaging using model-dependent weights, similar in concept to the popular multi-head self-attention operator in existing transformer networks. The overall gradient step (5.2.16) resembles the multi-head self-attention implemented with a skip connection in transformers.

MLP as proximal gradient descent for sparse coding of token representations. For the second step of alternating minimization, we need to minimize $\lambda\|\boldsymbol{Z}\|_1 - R_\epsilon(\boldsymbol{Z})$. Note that the gradient $\nabla R_\epsilon(\boldsymbol{Z})$ involves a matrix inverse, and thus naive proximal gradient (see Section A.1.3) to optimize this problem becomes intractable on large-scale problems. We therefore take a different approach to trading off between representational diversity and sparsification: we posit a (complete) incoherent or orthogonal dictionary $\boldsymbol{D}^\ell \in \mathbb{R}^{d\times d}$, and ask to sparsify the intermediate iterates $\boldsymbol{Z}^{\ell+1/2}$ with respect to $\boldsymbol{D}^\ell$. That is, $\boldsymbol{Z}^{\ell+1/2} \approx \boldsymbol{D}^\ell \boldsymbol{Z}^{\ell+1}$ where $\boldsymbol{Z}^{\ell+1}$ is more sparse; that is, it is a *sparse encoding* of $\boldsymbol{Z}^{\ell+1/2}$. The dictionary $\boldsymbol{D}^\ell$ is used to sparsify all tokens simultaneously. By the incoherence assumption, we have $(\boldsymbol{D}^\ell)^\top(\boldsymbol{D}^\ell) \approx \boldsymbol{I}$. Thus from (5.2.2) we have

$$R_\epsilon(\boldsymbol{Z}^{\ell+1/2}) \approx R_\epsilon(\boldsymbol{D}^\ell \boldsymbol{Z}^{\ell+1}) \approx R_\epsilon(\boldsymbol{Z}^{\ell+1}). \tag{5.2.17}$$

To solve $\lambda\|\boldsymbol{Z}\|_1 - R_\epsilon(\boldsymbol{Z})$, we optimize the following problem

$$\boldsymbol{Z}^{\ell+1} \approx \arg\min_{\boldsymbol{Z}} \|\boldsymbol{Z}\|_1 \quad \text{subject to} \quad \boldsymbol{Z}^{\ell+1/2} = \boldsymbol{D}^\ell \boldsymbol{Z}.$$

The above sparse representation program is usually solved by relaxing it to an unconstrained convex program, known as LASSO [WM22]:

$$\boldsymbol{Z}^{\ell+1} \approx \arg\min_{\boldsymbol{Z}} \left[\lambda\|\boldsymbol{Z}\|_1 + \frac{1}{2}\|\boldsymbol{Z}^{\ell+1/2} - \boldsymbol{D}^\ell \boldsymbol{Z}\|_F^2\right]. \tag{5.2.18}$$

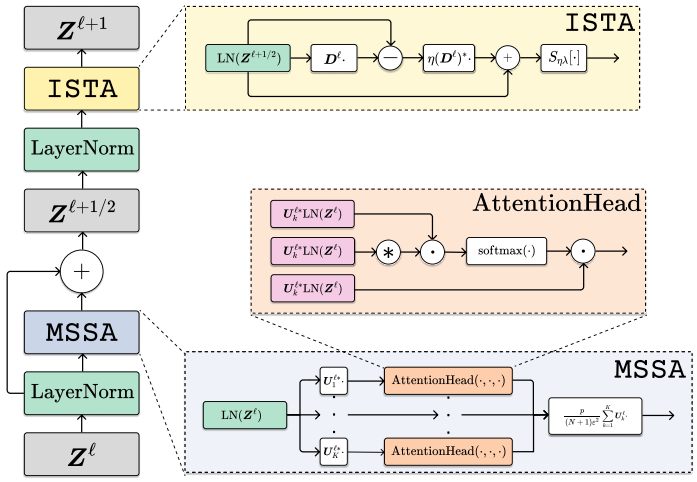


Figure 5.13: **One layer of the CRATE encoder architecture.** The full architecture is simply a concatenation of such layers, with some initial tokenizer, pre-processing head, and final task-specific head (i.e., a classification head).

In our implementation, we also add a non-negative constraint to $\boldsymbol{Z}^{\ell+1}$, and solve the corresponding non-negative LASSO:

$$\boldsymbol{Z}^{\ell+1} \approx \arg\min_{\boldsymbol{Z}\geq \boldsymbol{0}} \left[\lambda\|\boldsymbol{Z}\|_1 + \frac{1}{2}\|\boldsymbol{Z}^{\ell+1/2} - \boldsymbol{D}^\ell \boldsymbol{Z}\|_F^2\right]. \tag{5.2.19}$$

Then, we incrementally optimize Equation (5.2.19) by performing an unrolled *proximal gradient descent* step, known as an ISTA step [BT09], to give the update:

$$\boldsymbol{Z}^{\ell+1} = \mathrm{ISTA}(\boldsymbol{Z}^{\ell+1/2} \mid \boldsymbol{D}^\ell), \tag{5.2.20}$$

$$\text{where} \quad \mathrm{ISTA}(\boldsymbol{Z} \mid \boldsymbol{D}) \doteq \mathrm{ReLU}(\boldsymbol{Z} - \eta \boldsymbol{D}^\top(\boldsymbol{D}\boldsymbol{Z} - \boldsymbol{Z}) - \eta\lambda\boldsymbol{1}). \tag{5.2.21}$$

5.2.2 Overall White-Box Transformer Architecture: CRATE

We now design a white-box transformer architecture, named the Coding RATE Transformer (CRATE), by unrolling the above updates. By combining the above two steps (5.2.16) and (5.2.20):

1. Local compression of tokens within a sample towards a mixture-of-subspace structure, leading to the multi-head subspace self-attention block – MSSA;

2. Global sparsification of token sets across all samples through sparse coding, leading to the sparsification block – ISTA;

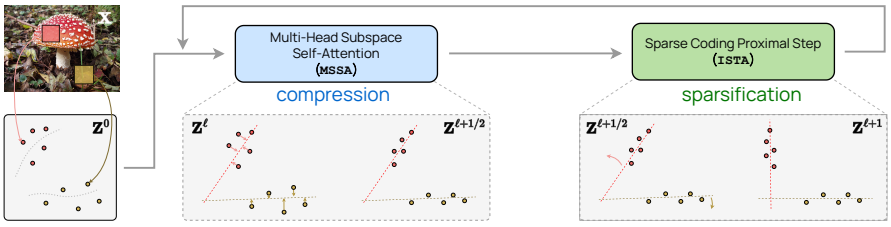


Figure 5.14: **The 'main loop' of the CRATE white-box deep network design.** After encoding input data as a sequence of tokens $\boldsymbol{Z}^0$, CRATE constructs a deep network that transforms the data to a canonical configuration of low-dimensional subspaces by successive ***compression*** against a local model for the distribution, generating $\boldsymbol{Z}^{\ell+1/2}$, and ***sparsification*** against a global dictionary, generating $\boldsymbol{Z}^{\ell+1}$. Repeatedly stacking these blocks and training the model parameters via backpropagation yields a powerful and interpretable representation of the data.

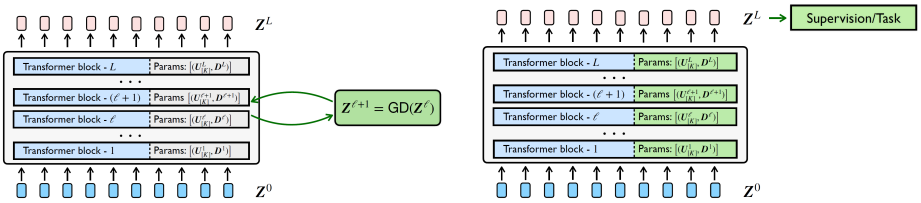
we can get the following rate-reduction-based transformer layer, illustrated in Figure 5.13,

$$\boldsymbol{Z}^{\ell+1/2} \doteq \boldsymbol{Z}^\ell + \mathrm{MSSA}(\boldsymbol{Z}^\ell \mid \boldsymbol{U}^\ell_{[K]}), \qquad \boldsymbol{Z}^{\ell+1} \doteq \mathrm{ISTA}(\boldsymbol{Z}^{\ell+1/2} \mid \boldsymbol{D}^\ell). \quad (5.2.22)$$

Composing multiple such layers following the incremental construction of our representation in (5.2.6), we obtain a white-box transformer architecture that transforms the data tokens towards a compact and sparse union of incoherent subspaces, where $f^{\text{pre}} : \mathbb{R}^{D\times N} \to \mathbb{R}^{d\times N}$ is the pre-processing mapping that transforms the input tokens $\boldsymbol{X} \in \mathbb{R}^{D\times N}$ to first-layer representations $\boldsymbol{Z}^1 \in \mathbb{R}^{d\times N}$. An overall flow of this architecture was shown in Figure 5.14.

Remark 5.8 (**The roles of the forward pass and backward propagation**). In contrast to other unrolled optimization approaches such as the ReduNet [CYY+22], we *explicitly model* the distribution of each $\boldsymbol{Z}^\ell$ and $\boldsymbol{Z}^{\ell+1/2}$ at each layer, either by a mixture of linear subspaces or sparsely generated from a dictionary. We introduced the interpretation that at each layer ℓ, the learned bases for the subspaces $\boldsymbol{U}^\ell_{[K]}$ and the learned dictionaries $\boldsymbol{D}^\ell$ together serve as a *codebook* or *analysis filter* that encodes and transforms the intermediate representations at each layer ℓ. Since the input distribution to layer ℓ is first modeled by $\boldsymbol{U}^\ell_{[K]}$ then transformed by $\boldsymbol{D}^\ell$, the input distribution to each layer is different, and so we require a separate codebook at each layer to obtain the most parsimonious encoding. Parameters of these codebooks (i.e., the subspace bases and the dictionaries), heretofore assumed as being perfectly known, are actually learned from data (say via *backward propagation* within end-to-end training).

Hence, our methodology features a clear conceptual separation between forward "optimization" and backward "learning" for the so-derived white-box deep neural network. Namely, in its forward pass, we interpret each layer as an operator which, conditioned on a learned model (i.e., a codebook) for the distri-



(a) **Forward pass** (b) **Backward propagation**

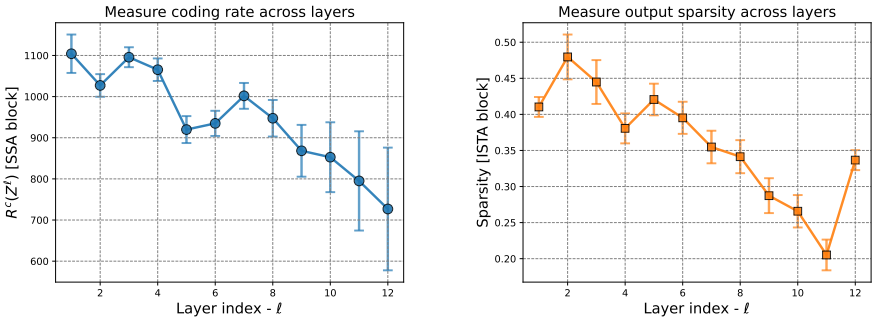
Figure 5.15: **The roles of forward pass and backward propagation in CRATE**. (a) Given fixed subspaces and dictionaries $\{(\boldsymbol{U}^\ell_{[K]}, \boldsymbol{D}^\ell)\}_{\ell=1}^L$, each layer incrementally optimizes the sparse rate reduction of the representations in the forward pass; (b) Backpropagation learn subspaces and dictionaries $\{(\boldsymbol{U}^\ell_{[K]}, \boldsymbol{D}^\ell)\}_{\ell=1}^L$ from training data.

bution of its input, transforms this distribution towards a more parsimonious representation. In its backward propagation, the codebook of this model, for the distribution of the input to each layer, is updated to better fit a certain (supervised) input-output relationship, as illustrated in Figure 5.15. This conceptual interpretation implies a certain agnosticism of the model representations towards the particular task and loss; in particular, many types of tasks and losses will ensure that the models at each layer are fit, which ensures that the model produces parsimonious representations.

There are three natural questions we can ask about CRATE:

- Do such constructed CRATE models learn good features by optimizing the sparse rate reduction?
- Do such constructed CRATE models use their high-quality features to perform well on predictive tasks?
- Do such constructed CRATE models exhibit interesting phenomena, say related to interpretability, as a result of their simplified design?

We contend that the answer to each question is "yes". First, Figure 5.16 examines how the network optimizes each term of the sparse rate reduction; each layer consistently decreases the features' compression measure and sparsity measures during the forward pass. Second, in Table 5.1 we demonstrate that CRATE has comparable top-1 classification accuracy under transfer learning setups to the most commonly used Vision Transformer (ViT) [DBK+21] neural network architecture at similar parameter counts, with promising scaling behavior — we obtain steady and consistent improvements by increasing the model size. Overall, CRATE achieves promising performance on real-world large-scale datasets by directly implementing our principled architecture. We provide more details of the implementation and analysis of the experimental results in Section 8.4.



(a) (b)

Figure 5.16: **Optimization of compression and sparsity measures through the forward pass of CRATE.** We measure the compression $R^c_\epsilon(\boldsymbol{Z}^{\ell+1/2})$ (a) and sparsity $\|\boldsymbol{Z}^{\ell+1}\|_1$ (b) at the output of each layer. We observe that both measures decrease over the course of the forward pass, as predicted by theory, and almost every layer monotonically decreases each measure. (The exception is the last layer's sparsity, since dense features are important for prediction.)

Table 5.1: **Top-1 classification accuracy of CRATE on various datasets with different model scales when pre-trained on ImageNet-1K.** For ImageNet-1K/ImageNet-1K ReaL, we directly evaluate the top-1 accuracy. For other datasets, we use models that are pre-trained on ImageNet as initialization and the evaluate the transfer learning performance via fine-tuning.

Model	CRATE-T	CRATE-S	CRATE-B	CRATE-L	ViT-T	ViT-S
# parameters	6.09M	13.12M	22.80M	77.64M	5.72M	22.05M
ImageNet-1K	66.7	69.2	70.8	71.3	71.5	72.4
ImageNet-1K ReaL	74.0	76.0	76.5	77.4	78.3	78.4
CIFAR10	95.5	96.0	96.8	97.2	96.6	97.2
CIFAR100	78.9	81.0	82.7	83.6	81.8	83.2
Oxford Flowers-102	84.6	87.1	88.7	88.3	85.1	88.5
Oxford-IIIT-Pets	81.4	84.9	85.3	87.4	88.5	88.6

5.3 Variants of Deep Architectures by Design

So far, we wish that we have provided compelling evidence that the role of (popular) deep networks is to realize certain optimization algorithms for minimizing the coding rate (or maximizing the information gain) of the learned representations. However, readers who are familiar with optimization methods might have noticed that the above architectures (the ReduNet or the CRATE) correspond to rather basic optimization techniques. They may have plenty of room for improvement in efficiency or effectiveness. Moreover, if we believe the proposed theoretical framework for interpreting deep networks is correct, it should not only help explain existing architectures, it should guide us develop more efficient and effective architectures. In this section, we show this is the case: the resulting new architectures are not only fully interpretable but also

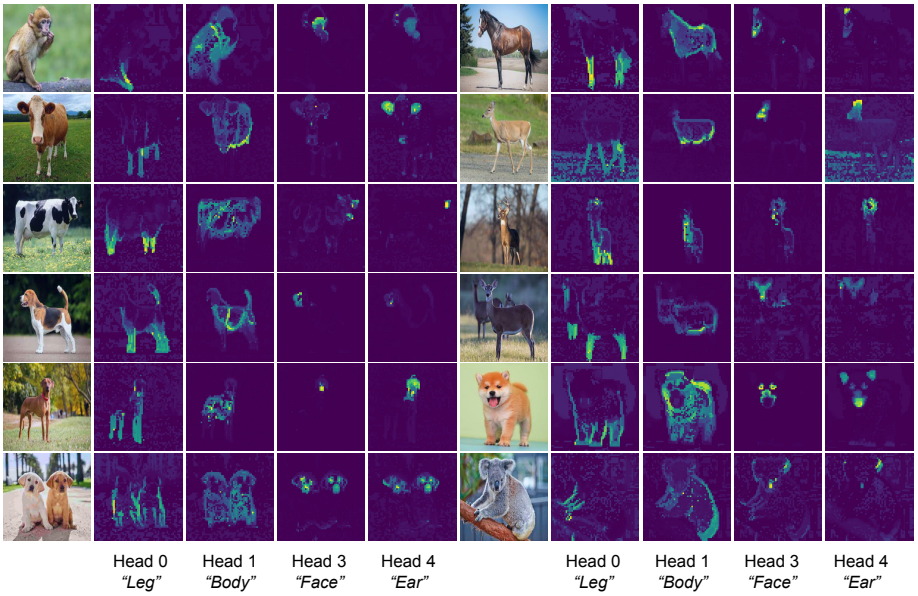
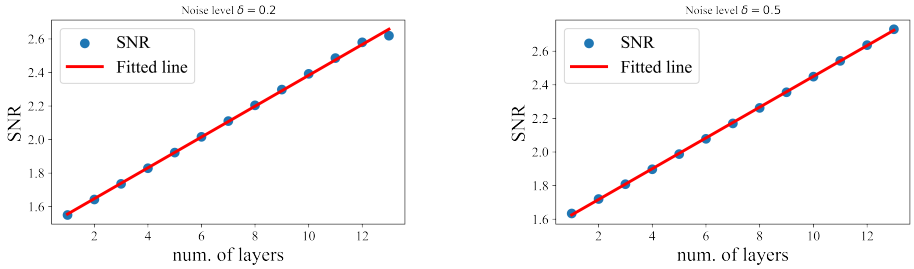


Figure 5.17: **Visualizing the attention maps of CRATE: what does the model look at to make a classification?** We visualize the last-layer attention maps, which are the last source of token-to-token interaction in the network, and therefore in some sense the most refined description of what sections of the input image the model uses to make a prediction. We observe an *extremely interesting* behavior: when trained on purely classification tasks, the attention maps identify the foreground objects in the image; moreover, they *naturally specialize* to different types of objects in the foreground! This empirical result gives us an interpretation of later-layer subspaces as each corresponding to a nearly-disjoint set of *concepts*.

with guaranteed correctness and improved efficiency.

5.3.1 Attention-Only Transformer Architecture

In this subsection, we ask and answer the question: "what is the simplest possible neural network which scales reasonably well and provably compresses the representation?" From our previous theory, the answer should be a stack of MSSA layers, which were motivated as lossy compression operator. This network architecture is simple enough such that it is possible to analyze the compression or denoising performance rigorously and systematically. Towards this end, let us formalize our subspace model: the initial token representations $\boldsymbol{Z}^1$ are sampled from a noisy mixture of low-rank Gaussians, supported on subspaces spanned by $\boldsymbol{U}_k \in \mathbb{R}^{d \times p}$, as follows: we partition the N token indices into



(a) Noise level $\delta = 0.2$ (b) Noise level $\delta = 0.5$

Figure 5.18: **Denoising performance of the attention-only transformer.** Here, we sample initial token representations from a mixture of low-rank Gaussians in Equation (5.3.1). Then, we apply (5.3.2) to update token representations and report the SNR at each layer.

known subsets $C_1, \ldots, C_K$, such that:

$$\boldsymbol{z}_i = \underbrace{\boldsymbol{U}_k \boldsymbol{a}_i}_{signal} + \underbrace{\sum_{j \neq k}^{K} \boldsymbol{U}_j \boldsymbol{e}_{i,j}}_{noise}, \ \forall i \in C_k, \tag{5.3.1}$$

where $\boldsymbol{a}_i \overset{i.i.d.}{\sim} \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_{p_k})$ and $\boldsymbol{e}_{i,j} \overset{i.i.d.}{\sim} \mathcal{N}(\boldsymbol{0}, \delta^2 \boldsymbol{I}_{p_j})$ for all $i \in C_k$ and $k \in [K]$, $\{\boldsymbol{a}_i\}$ and $\{\boldsymbol{e}_{i,j}\}$ are all independent.

Denoising operator for token representations. Now, we show that the MSSA operator (see (5.2.15)) can incrementally denoise token representations generated from the above model. Specifically, we consider a generalized MSSA operator: for each $\ell \in [L]$,

$$\boldsymbol{Z}^{\ell+1} = \boldsymbol{Z}^{\ell} + \eta \sum_{k=1}^{K} \boldsymbol{U}_k \boldsymbol{U}_k^T \boldsymbol{Z}^{\ell} \varphi(\boldsymbol{Z}^{\ell^T} \boldsymbol{U}_k \boldsymbol{U}_k^T \boldsymbol{Z}^{\ell}), \tag{5.3.2}$$

where $\eta > 0$ is the step size, and φ is a function operating on each column of the input matrix, such as softmax, element-wise ReLU, etc. To simplify our theory, we assume that the subspaces in Equation (5.3.1) are orthogonal to each other, i.e., $\boldsymbol{U}_k^T \boldsymbol{U}_j = \boldsymbol{0}$ for all $k \neq j$. Note that this assumption is not restrictive, as in high-dimensional spaces, random low-dimensional subspaces are incoherent to each other with high probability, i.e., $\boldsymbol{U}_k^T \boldsymbol{U}_j \approx \boldsymbol{0}$ [WM22].[19]

Now, let the columns of $\boldsymbol{Z}_k^{\ell}$ denote the token representations from the k-th subspace at the ℓ-th layer. To quantify the denoising capability, we define the

[19] One may straightforwardly generalize our results to non-orthogonal subspaces, with slightly more sophisticated analysis.

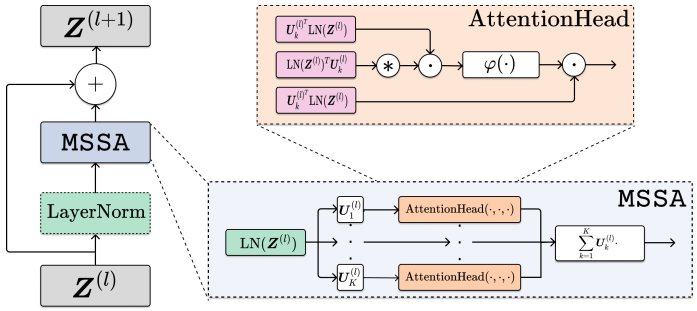


Figure 5.19: **Details of the attention-only transformer architecture.** Each layer consists of the MSSA operator and a skip connection. In addition, LayerNorm is included only for language tasks. In practice, backpropagation is applied to train the model parameters using training samples.

signal-to-noise ratio (SNR) for each block of the token representations at the ℓ-th layer as follows:

$$\mathrm{SNR}(\boldsymbol{Z}_k^\ell) \doteq \frac{\|\boldsymbol{U}_k\boldsymbol{U}_k^T\boldsymbol{Z}_k^\ell\|_F}{\|(\boldsymbol{I}-\boldsymbol{U}_k\boldsymbol{U}_k^T)\boldsymbol{Z}_k^\ell\|_F}, \quad \forall k \in [K]. \tag{5.3.3}$$

To simplify our analysis, we assume that $|C_1| = \cdots = |C_K| = N/K$, and

$$\begin{bmatrix}\boldsymbol{U}_1 & \ldots & \boldsymbol{U}_K\end{bmatrix} \in \mathcal{O}^{d\times Kp}. \tag{5.3.4}$$

With the above setup, we can now characterize the denoising performance of a slightly modified MSSA operator, which thresholds terms which are too small in the softmax.

Theorem 5.1. *Let $\boldsymbol{Z}^1$ be defined in Equation (5.3.1) and $\varphi(\cdot)$ in (5.3.2) be $\varphi(\boldsymbol{x}) = h(\sigma(\boldsymbol{x}))$, where $\sigma : \mathbb{R}^N \to \mathbb{R}^N$ is the softmax function and $h : \mathbb{R}^N \to \mathbb{R}^N$ is an element-wise thresholding function with $h(x) = \tau$ if $x > \tau$ and $h(x) = 0$ if $x \le \tau$ for each $i \in [N]$. Suppose that $p \gtrsim \log N$, $\delta \lesssim \sqrt{\log N}/\sqrt{p}$, and*

$$\tau \in \left(\frac{1}{2}, \frac{1}{1+N\exp(-9p/32)}\right].$$

For sufficiently large N, it holds with probability at least $1 - KLN^{-\Omega(1)}$ that for each $\ell \in [L]$,

$$\mathrm{SNR}(\boldsymbol{Z}_k^{\ell+1}) = (1+\eta\tau)\mathrm{SNR}(\boldsymbol{Z}_k^\ell), \ \forall k \in [K]. \tag{5.3.5}$$

This theorem demonstrates that when the initial token representations are sampled from a mixture of low-rank Gaussian distributions with a noise level $O(\sqrt{\log N}/\sqrt{p})$, we show that each application of the modified MSSA operator denoises token representations at a linear rate. Notably, our theoretical

Table 5.2: **Evaluating AoT on vision tasks**: top-1 accuracy on ImageNet.

Models	Accuracy	# of Parameters
AoT	71.7%	22M
CRATE	79.5%	39M
ViT	72.4 %	22M

results are well-supported by experimental observations using the *actual* MSSA operator in Figure 5.18. This theorem provides a theoretical foundation for the practical denoising capability of the transformer architecture derived by unrolling (5.3.2).

Attention-only transformers. To verify that this denoising capability translates to reasonable performance on real data and tasks, we formally propose an attention-only transformer architecture, which we short-hand as AoT. Specifically, by unrolling the iterative optimization steps (5.3.2) as layers of a deep network, we construct a transformer architecture in Figure 5.19. Each layer of the proposed architecture only consists of the MSSA operator and a skip connection. For language tasks, we additionally incorporate LayerNorm before the MSSA operator to improve performance. The complete architecture is built by stacking such layers, along with essential task-specific pre-processing and post-processing steps, such as positional encoding, token embedding, and a final task-specific head to adapt to different applications. Table 5.2 and demonstrates the performance of this minimalist transformer on vision classification; the performance is reasonable, despite the minimalist design. In the sequel, we will discuss a conceptual improvement to the CRATE formulation which *also* improve the performance and efficiency.

5.3.2 Linear-Time Attention: Token Statistics Transformer

In this subsection, we propose a new transformer attention operator whose computational complexity scales linearly with the number of tokens based on the coding rate reduction objective. Specifically, we derive a novel variational form of the MCR^2 objective and show that the architecture that results from unrolled gradient descent of this variational objective leads to a new attention module called Token Statistics Self-Attention (TSSA). TSSA has *linear computational and memory complexity* and radically departs from the typical attention architecture that computes pairwise similarities between tokens. Replacing a traditional attention operator in a transformer with TSSA yields a new neural network architecture called the Token Statistics Transformer (ToST), which is highly efficient and has promising empirical performance. As a preliminary, recall from (4.2.17) that $\mathbf{\Pi} = [\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_K] \in \mathbb{R}^{N \times K}$ denotes a stochastic "group assignment" matrix (i.e., $\mathbf{\Pi}\mathbf{1} = \mathbf{1}$ and $\Pi_{ik} \geq 0,\ \forall (i,k) \in [N] \times [K]$), where Π_{ik} denotes the probability of assigning the i-th token to the k-th group.

A new variational form for coding rates. To begin, we consider a general form of MCR^2-like objectives based on concave functions of the spectrum of a matrix. Namely, for a given PSD matrix $\boldsymbol{M} \in \mathsf{PSD}(d)$ and any scalar $c \geq 0$ we have that $\log\det(\boldsymbol{I} + c\boldsymbol{M}) = \sum_{i=1}^{d} \log(1 + c\lambda_i(\boldsymbol{M}))$, where $\lambda_i(\boldsymbol{M})$ is the i-th largest eigenvalue of $\boldsymbol{M}$. Further, note that $\log(1 + c\sigma)$ is a concave non-decreasing function of σ. Thus, we describe our results in terms of a more general form of MCR^2 based on general spectral functions of PSD matrices of the form $F(\boldsymbol{M}) = \sum_{i=1}^{d} f(\lambda_i(\boldsymbol{M}))$, where f is concave and non-decreasing. In particular, recall from earlier in this Chapter that the attention mechanism arises from unrolling the compression component of MCR^2, so we consider a more general MCR^2-style compression function:

$$R_{c,f}(\boldsymbol{Z}, \boldsymbol{\Pi}) \doteq \frac{1}{2} \sum_{k=1}^{K} \frac{N_k}{N} F\left(\frac{1}{N_k} \boldsymbol{Z}\mathrm{Diag}(\boldsymbol{\pi}_k)\boldsymbol{Z}^\top\right). \tag{5.3.6}$$

For the above objective, we now note the following result:

Theorem 5.2. *Let $f\colon [0, \infty) \to \mathbb{R}$ be non-decreasing, concave, and obey $f(0) = 0$, and let $F\colon \mathsf{PSD}(d) \to \mathbb{R}$ have the form $F(\boldsymbol{M}) = \sum_{i=1}^{d} f(\lambda_i(\boldsymbol{M}))$. Then for each $\boldsymbol{M} \in \mathsf{PSD}(d)$ and $\boldsymbol{Q} \in \mathsf{O}(d)$, we have*

$$F(\boldsymbol{M}) \leq \sum_{i=1}^{d} f\left((\boldsymbol{Q}^\top \boldsymbol{M} \boldsymbol{Q})_{ii}\right). \tag{5.3.7}$$

Further, the inequality in (5.3.7) is achieved with equality for any $\boldsymbol{Q}$ which diagonalizes $\boldsymbol{M}$, and if f is strictly concave then the inequality in (5.3.7) is achieved with equality if and only if $\boldsymbol{Q}$ diagonalizes $\boldsymbol{M}$.

Using the above result, we can replace (5.3.6) with an equivalent variational objective with form

$$R^{\mathrm{var}}_{c,f}(\boldsymbol{Z}, \boldsymbol{\Pi} \mid \boldsymbol{U}_{[K]}) \doteq \frac{1}{2} \sum_{k=1}^{K} \frac{N_k}{N} \sum_{i=1}^{d} f\left(\frac{1}{N_k} (\boldsymbol{U}_k^\top \boldsymbol{Z}\mathrm{Diag}(\boldsymbol{\pi}_k)\boldsymbol{Z}^\top \boldsymbol{U}_k)_{ii}\right), \tag{5.3.8}$$

where the equivalence is in the sense that for an optimal choice of $\{\boldsymbol{U}_k \in \mathsf{O}(d)\}_{k=1}^{K}$ matrices as described in Theorem 5.2 (i.e., orthogonal matrices which diagonalize each $\boldsymbol{Z}\mathrm{Diag}(\boldsymbol{\pi}_k)\boldsymbol{Z}^\top$) we will achieve a tight bound with $R^{\mathrm{var}}_{c,f}(\boldsymbol{Z}, \boldsymbol{\Pi} \mid \boldsymbol{U}_{[K]}) = R_{c,f}(\boldsymbol{Z}, \boldsymbol{\Pi})$. Note that in general, achieving this bound would require selecting, for each sampled instance of $\boldsymbol{Z}$, a new optimal set of $\boldsymbol{U}_k$ parameter matrices which diagonalize each $\boldsymbol{Z}\mathrm{Diag}(\boldsymbol{\pi}_k)\boldsymbol{Z}^\top$, which is clearly impractical for network architecture. Instead, as an alternative viewpoint, rather than considering the data ($\boldsymbol{Z}$) as fixed and trying to optimize the $\boldsymbol{U}_k$ parameters to achieve the tight variational bound, we can instead take the algorithmic unrolling design principle described above and design an operator to perturb $\boldsymbol{Z}$ to incrementally minimize $R^{\mathrm{var}}_{c,f}(\cdot \mid \boldsymbol{U}_{[K]})$. To make this point explicit, each variational bound becomes tight when the eigenspaces of $\boldsymbol{Z}\mathrm{Diag}(\boldsymbol{\pi}_k)\boldsymbol{Z}^\top$ align with

the columns of $\boldsymbol{U}_k$, so by rotating the appropriate columns of $\boldsymbol{Z}$ (namely, those which correspond to large entries in $\boldsymbol{\pi}_k$) to align with $\boldsymbol{U}_k$ we can approach a tight variational bound. That is, instead of rotating $\boldsymbol{U}_k$ to align with the data for each instance of $\boldsymbol{Z}$, we can instead rotate the token features in each $\boldsymbol{Z}$ to align with $\boldsymbol{U}_k$.

Following this approach, we compute a gradient descent step on $R^{\text{var}}_{c,f}$ w.r.t. $\boldsymbol{Z}$. To begin this computation, first let $\boldsymbol{\pi} \in \mathbb{R}^N$ be any element-wise non-negative vector. Then we have

$$\nabla_{\boldsymbol{Z}} \frac{1}{2}\sum_{i=1}^{d} f((\boldsymbol{Z}\mathrm{Diag}(\boldsymbol{\pi})\boldsymbol{Z}^\top)_{ii}) = \mathrm{Diag}(\nabla f[\boldsymbol{Z}^{\circ 2}\boldsymbol{\pi}])\boldsymbol{Z}\mathrm{Diag}(\boldsymbol{\pi}), \tag{5.3.9}$$

where ∇f is the gradient of f, and (recall) $\nabla f[\cdot]$ applies ∇f to each element of the vector in the bracket. In particular, for $f(x) = \log(1 + (d/\epsilon^2)x)$, $\nabla f(x) = (d/\epsilon^2)(1 + (d/\epsilon^2)x)^{-1}$ is simply a non-linear activation. Also, (recall) $N_k = \langle \boldsymbol{\pi}_k, \mathbf{1}\rangle$. Thus, the gradient of $R^{\text{var}}_{c,f}$ w.r.t. $\boldsymbol{Z}$ is:

$$\nabla_{\boldsymbol{Z}} R^{\text{var}}_{c,f}(\boldsymbol{Z}, \boldsymbol{\Pi} \mid \boldsymbol{U}_{[K]}) \tag{5.3.10}$$

$$= \frac{1}{n}\sum_{k=1}^{K} \boldsymbol{U}_k \underbrace{\mathrm{Diag}\left(\nabla f\left[(\boldsymbol{U}_k^\top \boldsymbol{Z})^{\circ 2}\frac{\boldsymbol{\pi}_k}{\langle \boldsymbol{\pi}_k, \mathbf{1}\rangle}\right]\right)}_{\doteq \boldsymbol{D}(\boldsymbol{Z}, \boldsymbol{\pi}_k \mid \boldsymbol{U}_k)} \boldsymbol{U}_k^\top \boldsymbol{Z}\mathrm{Diag}(\boldsymbol{\pi}_k). \tag{5.3.11}$$

(Note that the $1/N$ constant arises from a $(N_k/N)\cdot(1/N_k) = 1/N$ constant in each term of the sum.) If we now consider a gradient step w.r.t. the j-th token $\boldsymbol{z}_j$, we arrive at our proposed incremental compression operator, i.e., our surrogate for a *self attention* + residual operator:

$$\boldsymbol{z}_j^+ = \boldsymbol{z}_j - \tau \nabla_{\boldsymbol{z}_j} R^{\text{var}}_{c,f}(\boldsymbol{Z}, \boldsymbol{\Pi} \mid \boldsymbol{U}_{[K]}) \tag{5.3.12}$$

$$= \boldsymbol{z}_j - \frac{\tau}{N}\sum_{k=1}^{K} \Pi_{jk}\boldsymbol{U}_k \boldsymbol{D}(\boldsymbol{Z}, \boldsymbol{\pi}_k \mid \boldsymbol{U}_k)\boldsymbol{U}_k^\top \boldsymbol{z}_j \tag{5.3.13}$$

for each $j \in [n]$, where $\tau > 0$ is a step size parameter for the incremental optimization.

Model interpretation. Given the proposed attention operator in (5.3.12), first recall that the rows of $\boldsymbol{\Pi}$ are non-negative and sum to 1 , so our operator takes a weighted average of K "attention head"-esque operators and then adds a residual connection. Using that $\sum_{k=1}^{K} \Pi_{jk} = 1$, we can rewrite (5.3.12) as:

$$\boldsymbol{z}_j^+ = \sum_{k=1}^{K} \Pi_{jk}\Big[\boldsymbol{z}_j \underbrace{- \frac{\tau}{n}\boldsymbol{U}_k \boldsymbol{D}(\boldsymbol{Z}, \boldsymbol{\pi}_k \mid \boldsymbol{U}_k)\boldsymbol{U}_k^\top \boldsymbol{z}_j}_{\text{action of one attention head}}\Big]. \tag{5.3.14}$$

That is, we can view each attention head as first projecting the token features onto the basis $\boldsymbol{U}_k$ via multiplying by $\boldsymbol{U}_k^\top$, multiplying by the diagonal matrix

$\boldsymbol{D}(\boldsymbol{Z}, \boldsymbol{\pi}_k \mid \boldsymbol{U}_k)$ (abbreviated as $\boldsymbol{D}_k$), projecting back into the standard basis via multiplying by $\boldsymbol{U}_k$, and subtracting this from the original token features via the residual connection. The core aspect of our attention layer is the computation of $\boldsymbol{D}_k$. Namely, $\Pi_{jk} \geq 0$, so $\boldsymbol{\pi}_k / \langle \boldsymbol{\pi}_k, \mathbf{1} \rangle \in \mathbb{R}^N$ forms a probability distribution over which tokens belong to the k^{th} group. As a result, $(\boldsymbol{U}_k^\top \boldsymbol{Z})^{\circ 2} \boldsymbol{\pi}_k / \langle \boldsymbol{\pi}_k, \mathbf{1} \rangle$ estimates the second moment of $\boldsymbol{U}_k^\top \boldsymbol{Z}$ under the distribution given by $\boldsymbol{\pi}_k / \langle \boldsymbol{\pi}_k, \mathbf{1} \rangle$. Further, since f is a concave non-decreasing function, $\nabla f(x)$ monotonically decreases towards 0 as x increases, so the entries of $\boldsymbol{D}_k$ (which have form $\nabla f(x)$) achieve their maximum at $x = 0$ and decay monotonically to 0 as x increases.

From this, we arrive at the core interpretation of our attention head + residual operators $[\boldsymbol{I} - (\tau/n)\boldsymbol{U}_k \boldsymbol{D}_k \boldsymbol{U}_k^\top]$. Namely, this operator does an approximate low-rank data-dependent projection, where directions which have a large amount of "power" after the projection $\boldsymbol{U}_k^\top \boldsymbol{Z}$ (i.e., directions which have a large second moment $(\boldsymbol{U}_k^\top \boldsymbol{Z})^{\circ 2} \boldsymbol{\pi}_k / \langle \boldsymbol{\pi}_k, \mathbf{1} \rangle$) are preserved, while directions which do not are suppressed. To see this, recall that the entries of $\boldsymbol{D}_k$ decrease monotonically to 0 as the second moment increases, so for directions with large second moments the attention + residual operator acts largely as the identity operator. Conversely, for directions with a small second moment, our operator subtracts a projection of the tokens along those directions, resulting in those directions being suppressed. Compared to the standard self-attention operator, our method clearly does not compute any pairwise similarities between tokens. Rather, the interactions between the tokens in $\boldsymbol{Z}$ impact the operator solely through their contribution to the second moment statistic used to construct the $\boldsymbol{D}_k$'s. Nevertheless, similar to the standard self-attention operator, our method still has a clear interpretation as performing a form of compression towards a data-dependent low-rank structure, in the sense that it performs an approximate low-rank projection, where the specific directions that are suppressed are those which are not strongly aligned with other tokens in the group.

Computational considerations. Having introduced our proposed attention operator, we now discuss how it can be computed practically. First, until this point in the presentation, we have avoided discussion of how tokens are "grouped" into various attention heads via the $\boldsymbol{\Pi}$ matrix, but clearly a means of constructing $\boldsymbol{\Pi}$ is needed to implement our method. Additionally, our variational form in Theorem 5.2 requires the $\boldsymbol{U}$ matrices to be square and orthogonal, but one would ideally like to use smaller matrices (i.e., reduce the number of columns in $\boldsymbol{U}$) for efficiency as well as drop the orthogonality constraints.

In practice, we do not enforce the orthogonality constraints. To reduce the number of columns in the $\boldsymbol{U}$ matrices, we note that similar to CRATE [YBP+23], if we assume the features $\boldsymbol{Z}$ within group k are (approximately) clustered around a low-dimensional subspace — say of dimension p — then the within-group-k covariance $\boldsymbol{Z}\mathrm{Diag}(\boldsymbol{\pi}_k)\boldsymbol{Z}^\top$ is low-rank, where recall that [YCY+20] shows that the optimal geometry of $\boldsymbol{Z}$ will be for each group to be a low-rank subspace, orthogonal to the other groups. We can thus explicitly find a low-dimensional orthonormal basis for the image of this covariance, i.e.,

the linear span of the data in group k. If the dimension is $p \leq d$, the basis can be represented by a $d \times p$ orthogonal matrix $\boldsymbol{U}_k \in \mathsf{O}(d,p)$. In this case, we can more efficiently upper-bound $R_{c,f}$ using these low-rank orthogonal basis matrices. To show this, we use a more general version of Theorem 5.2 to yield the following corollary.

Corollary 5.1. *Let $f\colon [0,\infty) \to \mathbb{R}$ be non-decreasing, concave, and obey $f(0) = 0$, and let $F\colon \mathsf{PSD}(p) \to \mathbb{R}$ have the form $F(\boldsymbol{M}) = \sum_{i=1}^{p} f(\lambda_i(\boldsymbol{M}))$. Let $\boldsymbol{Z}$, $\boldsymbol{\Pi}$ be fixed. Then, for all $\boldsymbol{U}_1, \ldots, \boldsymbol{U}_K \in \mathsf{O}(d,p)$ such that $\operatorname{image}(\boldsymbol{Z}\operatorname{diag}(\boldsymbol{\pi}_k)\boldsymbol{Z}^\top) \subset \operatorname{image}(\boldsymbol{U}_k)$ for all $k \in [K]$, we have*

$$R_{c,f}(\boldsymbol{Z}, \boldsymbol{\Pi}) \leq R^{\mathrm{var}}_{c,f}(\boldsymbol{Z}, \boldsymbol{\Pi} \mid \boldsymbol{U}_{[K]}), \tag{5.3.15}$$

where $R^{\mathrm{var}}_{c,f}$ is formally defined in (5.3.8). Equality holds if $\boldsymbol{U}_k$ diagonalizes $\boldsymbol{Z}\operatorname{diag}(\boldsymbol{\pi}_k)\boldsymbol{Z}^\top$ for each $k \in [K]$, and if f is strongly concave then this equality condition becomes an "if and only if."

The final step to define our attention operator is to estimate the group membership $\boldsymbol{\Pi}$. For this we posit a simple model of how each feature $\boldsymbol{z}_j$ deviates from its supporting subspace and then find the optimal subspace assignment. [YBP+23] show that if we independently model each $\boldsymbol{z}_j$ as belonging to a low-dimensional Gaussian mixture model, where each Gaussian has a covariance matrix with identical spectrum and is supported on a subspace with orthonormal basis $\boldsymbol{U}_k$, plus independent Gaussian noise with covariance $\eta\boldsymbol{I}$, then the posterior probability that each token $\boldsymbol{z}_j$ belongs to each subspace is given by the assignment matrix $\boldsymbol{\Pi} = \boldsymbol{\Pi}(\boldsymbol{Z} \mid \boldsymbol{U}_{[K]})$ as follows:

$$\boldsymbol{\Pi} = \begin{bmatrix} \boldsymbol{\nu}(\boldsymbol{z}_1 \mid \boldsymbol{U}_{[K]})^\top \\ \vdots \\ \boldsymbol{\nu}(\boldsymbol{z}_n \mid \boldsymbol{U}_{[K]})^\top \end{bmatrix}, \quad \text{where} \tag{5.3.16}$$

$$\boldsymbol{\nu}(\boldsymbol{z}_j \mid \boldsymbol{U}_{[K]}) \doteq \operatorname{softmax}\left(\frac{1}{2\eta} \begin{bmatrix} \|\boldsymbol{U}_1^\top \boldsymbol{z}_j\|_2^2 \\ \vdots \\ \|\boldsymbol{U}_K^\top \boldsymbol{z}_j\|_2^2 \end{bmatrix} \right), \quad \forall j \in [n], \tag{5.3.17}$$

where η becomes a learnable temperature parameter. Thus, given an input feature $\boldsymbol{Z}$, we estimate $\boldsymbol{\Pi}$ using (5.3.16) and then compute the attention operator. Combining the construction of $\boldsymbol{\Pi}$ in (5.3.16) with (5.3.12), we obtain the *Token Statistics Self-Attention* operator:

$$\operatorname{TSSA}(\boldsymbol{Z} \mid \boldsymbol{U}_{[K]}) \doteq -\frac{\tau}{n} \sum_{k=1}^{K} \boldsymbol{U}_k \boldsymbol{D}(\boldsymbol{Z}, \boldsymbol{\pi}_k \mid \boldsymbol{U}_k) \boldsymbol{U}_k^\top \boldsymbol{Z} \operatorname{diag}(\boldsymbol{\pi}_k), \tag{5.3.18}$$

where $\boldsymbol{\pi}_k$ are the columns of $\boldsymbol{\Pi} = \boldsymbol{\Pi}(\boldsymbol{Z} \mid \boldsymbol{U}_{[K]})$ defined in (5.3.16) and $\boldsymbol{D}$ is defined in (5.3.10).

By starting with a standard Transformer architecture and replacing the attention operator with TSSA, we obtain an architecture called Token Statistics

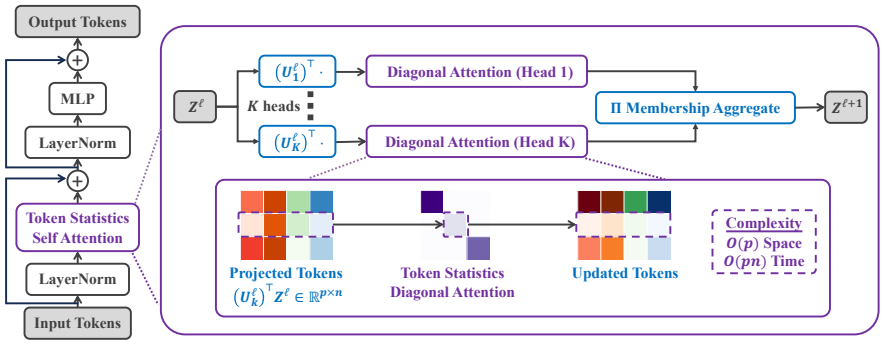
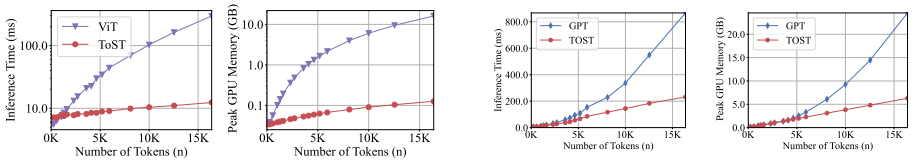


Figure 5.20: **One layer ℓ of the proposed Token Statistics Transformer (ToST).** Notably, the self-attention of ToST transforms tokens $\boldsymbol{Z}^\ell$ efficiently to $\boldsymbol{Z}^{\ell+1}$, via multiplying each row of the projected token by *only a scalar*. This leads to reduced complexity of the attention: it has $O(p)$ space and $O(pn)$ time complexity, where p is the dimension of the projected tokens of each head, and n is the number of tokens.



(a) (b)

Figure 5.21: **Time and memory complexity of ToST in practice** compared to empirically designed ViT (a) and GPT-2 (b). We observe that ToST only requires linear time and memory, in comparison to quadratic costs of traditional transformers.

Transformer (ToST), visualized in Figure 5.20. We would like to re-emphasize that the TSSA operator mitigates the usual time and memory complexity of the attention operator, in particular reducing the quadratic time and memory complexity to linear, which makes ToST much more efficient than its empirically designed counterpart. We now briefly show the empirical performance and efficiency of the ToST architecture.

5.3.3 Causal CRATE

Some of the most popular applications of deep networks are in the regimes of *sequence data* where, unlike the previous example of imagery, there is a natural order to the data. For example, unlike in imagery (where one tends to look at the whole image at once), language is processed one token at a time (causally). This simple fact motivates the development of large language models (LLMs), which are trained to predict the (probability distribution of) the next token in a sequence given *only the history of previous tokens*. Then, to sample from

Table 5.3: **Top 1 accuracy of ToST on various datasets with different model sizes when pre-trained on ImageNet.** For ImageNet/ImageNetReaL, we directly evaluate the top-1 accuracy. For other datasets, we pre-train the model on ImageNet and then evaluate the transfer learning performance via fine-tuning. We observe that ToST performance scales with model size and is competitive with the empirically designed ViT.

Datasets	ToST-T(iny)	ToST-S(mall)	ToST-M(edium)	ViT-S	ViT-B(ase)
# parameters	5.8M	22.6M	68.1M	22.1M	86.6 M
ImageNet	67.3	77.9	80.3	79.8	81.8
ImageNet ReaL	72.2	84.1	85.6	85.6	86.7
CIFAR10	95.5	96.5	97.5	98.6	98.8
CIFAR100	78.3	82.7	84.5	88.8	89.3
Oxford Flowers-102	88.6	92.8	94.2	94.0	95.7
Oxford-IIIT-Pets	85.6	91.1	92.8	92.8	94.1

Table 5.4: **Long-context performance of ToST.** We demonstrate performance on the Long Range Arena, a benchmark of long-context (large-n) tasks. On this benchmark, ToST has outright superior performance and efficiency compared to every Transformer variant, including the Transformer itself.

Model	ListOps	Text	Retrieval	Image	Pathfinder	Avg
Reformer	37.27	56.10	53.40	38.07	68.50	50.56
BigBird	36.05	64.02	59.29	40.83	74.87	54.17
LinFormer	16.13	65.90	53.09	42.34	75.30	50.46
Performer	18.01	65.40	53.82	42.77	77.05	51.18
Transformer	37.11	65.21	79.14	42.94	71.83	59.24
ToST	37.25	66.75	79.46	46.62	69.41	59.90

an LLM, one simply iteratively samples from the predicted distribution for the next token and then appends the sampled token to the history. In order to train LLMs, and other models such as video generation models, it is necessary to develop architectures which can efficiently perform this causal computation.

With this in mind, and the understanding of causal autoregressive processes from Chapter 1, we say that an encoder f is *causal* if for every input $\boldsymbol{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_N]$, it holds

$$f(\boldsymbol{X})_{1:N-1} = f([\boldsymbol{x}_1, \dots, \boldsymbol{x}_N])_{1:N-1} = f([\boldsymbol{x}_1, \dots, \boldsymbol{x}_{N-1}]) \tag{5.3.19}$$

$$= f(\boldsymbol{X}_{1:N-1}), \tag{5.3.20}$$

where $\boldsymbol{Z}_{1:N-1} = [\boldsymbol{z}_1, \dots, \boldsymbol{z}_{N-1}]$, etc. In English, this means that the first $N-1$ features are equal to the output of the encoder on the first $N-1$ token embeddings of the input; even simpler, it means we can compute the features of each token *one at a time*, and this would be equivalent to computing the features for the entire input. This causality is necessary for the previously mentioned applications (e.g., LLMs). In particular, the previously presented implementations of CRATE and ToST are *not causal* (proof is left as an exercise).

We can construct causal white-box architectures by a variety of methods. Here, we will showcase a simple method which builds on our previous unrolled optimization framework. Specifically, we compute the features $\boldsymbol{z}_1, \dots, \boldsymbol{z}_N$ corresponding to the first N tokens of the input *one-at-a-time* to optimize the representation learning objective, such as the sparse rate reduction (5.2.5):

$$\max_{\boldsymbol{z}_i} \Delta R(\boldsymbol{Z}_{1:i} \mid \boldsymbol{U}_{[K]}) - \lambda \|\boldsymbol{Z}_{1:i}\|_1, \ \forall i \in [N]. \tag{5.3.21}$$

If we follow through with the two-step unrolling procedure that yielded CRATE, we can obtain the iteration:

$$\boldsymbol{z}_i^{\ell+1/2} \approx \boldsymbol{z}_i^{\ell} - \kappa \nabla_{\boldsymbol{z}_i} R(\boldsymbol{Z}_{1:i}^{\ell} \mid \boldsymbol{U}_{[K]}^{\ell}), \tag{5.3.22}$$

$$\boldsymbol{z}_i^{\ell+1} \approx \arg\min_{\boldsymbol{z}_i} \left\{ \lambda \|\boldsymbol{z}_i\|_1 + \frac{1}{2} \|\boldsymbol{z}_i^{\ell+1/2} - \boldsymbol{D}^{\ell} \boldsymbol{z}_i\|_F^2 \right\}, \tag{5.3.23}$$

or, using the same conversion from these two steps to network operators that we used for CRATE,

$$\boldsymbol{z}_i^{\ell+1/2} = \left(1 - \frac{\kappa p}{N\epsilon^2}\right) \boldsymbol{z}_i^{\ell} + \frac{\kappa p}{N\epsilon^2} \operatorname{MSSA}(\boldsymbol{Z}_{1:i}^{\ell} \mid \boldsymbol{U}_{[K]}^{\ell})_i, \tag{5.3.24}$$

$$\boldsymbol{z}_i^{\ell+1} = \operatorname{ISTA}(\boldsymbol{z}_i^{\ell+1/2} \mid \boldsymbol{D}^{\ell}). \tag{5.3.25}$$

Let us investigate this iteration in slightly more detail. First, let us note that by construction, this sequence of features corresponds to a causal encoder. Next, let us suppose that we are in a setting where we have computed $\boldsymbol{Z}_{1:i-1}^{\ell+1} = [\boldsymbol{z}_1^{\ell+1}, \dots, \boldsymbol{z}_{i-1}^{\ell+1}]$, having computed the quantities

$$(\boldsymbol{U}_k^{\ell})^\top \boldsymbol{Z}_{1:i-1}^{\ell} = [(\boldsymbol{U}_k^{\ell})^\top \boldsymbol{z}_1^{\ell}, \dots, (\boldsymbol{U}_k^{\ell})^\top \boldsymbol{z}_{i-1}^{\ell}] \tag{5.3.26}$$

along the way, and want to compute $\boldsymbol{z}_i^{\ell+1}$ (for instance relevant to the case of LLM inference). In this case, note that the update rule for $\boldsymbol{z}_i^{\ell+1/2}$ can be simplified as:

$$\boldsymbol{z}_i^{\ell+1/2} \tag{5.3.27}$$

$$= \left(1 - \frac{\kappa p}{N\epsilon^2}\right) \boldsymbol{z}_i^{\ell} + \frac{\kappa p}{N\epsilon^2} \operatorname{MSSA}(\boldsymbol{Z}_{1:i}^{\ell} \mid \boldsymbol{U}_{[K]}^{\ell})_i \tag{5.3.28}$$

$$= \left(1 - \frac{\kappa p}{N\epsilon^2}\right) \boldsymbol{z}_i^{\ell} + \frac{\kappa p^2}{N^2\epsilon^4} \sum_{k=1}^{K} \boldsymbol{U}_k \boldsymbol{U}_k^\top \boldsymbol{Z}_{1:i} \operatorname{softmax}((\boldsymbol{U}_k^\top \boldsymbol{Z}_{1:i})^\top (\boldsymbol{U}_k^\top \boldsymbol{z}_i)) \tag{5.3.29}$$

$$= \left(1 - \frac{\kappa p}{N\epsilon^2}\right) \boldsymbol{z}_i^{\ell} + \frac{\kappa p^2}{N^2\epsilon^4} \sum_{k=1}^{K} \boldsymbol{U}_k \boldsymbol{U}_k^\top [\boldsymbol{Z}_{1:i-1}, \boldsymbol{z}_i] \operatorname{softmax}((\boldsymbol{U}_k^\top [\boldsymbol{Z}_{1:i-1}, \boldsymbol{z}_i])^\top (\boldsymbol{U}_k^\top \boldsymbol{z}_i)) \tag{5.3.30}$$

$$= \left(1 - \frac{\kappa p}{N\epsilon^2}\right) \boldsymbol{z}_i^{\ell} + \frac{\kappa p^2}{N^2\epsilon^4} \sum_{k=1}^{K} \boldsymbol{U}_k [\boldsymbol{U}_k^\top \boldsymbol{Z}_{1:i-1}, \boldsymbol{U}_k^\top \boldsymbol{z}_i] \operatorname{softmax}(([\boldsymbol{U}_k^\top \boldsymbol{Z}_{1:i-1}, \boldsymbol{U}_k^\top \boldsymbol{z}_i])^\top (\boldsymbol{U}_k^\top \boldsymbol{z}_i)). \tag{5.3.31}$$

This step now becomes *highly efficient* if we cache $\boldsymbol{U}_k^\top \boldsymbol{Z}_{1:i-1}$ from previous steps, and add a single column to it each time we compute the projection $\boldsymbol{U}_k^\top \boldsymbol{z}_i$. Namely, we greatly reduce the number of large matrix-matrix products and replace them by cache loads and matrix-vector products, which is overall much cheaper in terms of time complexity. This caching is the reason why causal generative models such as LLMs can efficiently sample 1000s of tokens per second, even if each training step takes a few seconds by itself. The cache for the subspace projections of the features is also known as the so-called "KV cache".

Finally, let us consider the case where we are to *train* a causal CRATE model, and want to find the most efficient way to do this given a full input sequence $\boldsymbol{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_N]$ simultaneously. The ISTA step is parallelizable to become the regular ISTA step from the non-causal CRATE, i.e.,

$$\mathrm{ISTA}(\boldsymbol{Z} \mid \boldsymbol{D}) \doteq [\mathrm{ISTA}(\boldsymbol{z}_1 \mid \boldsymbol{D}), \dots, \mathrm{ISTA}(\boldsymbol{z}_N \mid \boldsymbol{D})], \tag{5.3.32}$$

therefore implying that the ISTA step remains the same as the non-causal CRATE. The MSSA step is more interesting, since it changes in a meaningful way. To see how it changes, note that each MSSA operator has an interesting structure which merits some attention. Namely, if we define the *causal MSSA operator* as the block matrix:

$$\mathrm{CMSSA}(\boldsymbol{Z} \mid \boldsymbol{U}_{[K]}) \doteq [\mathrm{MSSA}(\boldsymbol{Z}_{1:1} \mid \boldsymbol{U}_{[K]})_1, \dots, \mathrm{MSSA}(\boldsymbol{Z}_{1:N} \mid \boldsymbol{U}_{[K]})_N], \tag{5.3.33}$$

then, working our way through the softmax algebra (proof is again left as an exercise), we can show that

$$\mathrm{CMSSA}(\boldsymbol{Z} \mid \boldsymbol{U}_{[K]}) \doteq \tfrac{p}{N\epsilon^2} [\boldsymbol{U}_1, \dots, \boldsymbol{U}_K] \begin{bmatrix} \mathrm{softmax}((\boldsymbol{U}_1^\top \boldsymbol{Z})^\top (\boldsymbol{U}_1^\top \boldsymbol{Z}) + \boldsymbol{M}_N) \\ \vdots \\ \mathrm{softmax}((\boldsymbol{U}_K^\top \boldsymbol{Z})^\top (\boldsymbol{U}_K^\top \boldsymbol{Z}) + \boldsymbol{M}_N) \end{bmatrix}, \tag{5.3.34}$$

$$\text{where } \boldsymbol{M}_N = \begin{bmatrix} 0 & -\infty & -\infty & \dots & -\infty & -\infty \\ 0 & 0 & -\infty & \dots & -\infty & -\infty \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & -\infty \\ 0 & 0 & 0 & \dots & 0 & 0 \end{bmatrix}. \tag{5.3.35}$$

Here the matrix $\boldsymbol{M}_N \in \mathbb{R}^{N \times N}$ is another way to encode that no feature in CMSSA, i.e., the causal MSSA, depends on a future feature, since the relevant entries of the argument to the softmax are $-\infty$, and thus are set to 0 after the exponential within the softmax, and therefore effectively ignored. This matrix $\boldsymbol{M}_N$ is sometimes called a *(causal) attention mask*.

The upshot of this is that when we have the full sequence $\boldsymbol{Z}^\ell$, we can compute $\boldsymbol{Z}^{\ell+1}$ in time similar to or less than that of the usual CRATE layer (since $\boldsymbol{M}_N$ is hard-coded, input-independent, and enables us to ignore many entries for the softmax). Thus, we can define and train the full sequence-to-sequence model in the same way as the regular architecture.

Table 5.5: Zero-shot validation cross-entropy loss of the Causal-CRATE-Base model and GPT2-Small, GPT2-Base model evaluated on the test split of the datasets (lower is better).

	#parameters	OWT	LAMBADA	WikiText	PTB	Avg
GPT2-Base	124M	2.85	4.12	3.89	4.63	3.87
GPT2-Small	64M	3.04	4.49	4.31	5.15	4.25
Causal-CRATE-Base	60M	3.37	4.91	4.61	5.53	4.61

When we apply this to language modeling (details to be provided in Chapter 8), we find that we can obtain reasonable results compared to similar-sized empirically designed language models, as shown in Table 5.5.

Despite the theoretical advantages of the causal CRATE architecture, the gap on language modeling (and more generally sequence modeling) tasks still remains. Below, we discuss two strategies for potentially closing this gap; they are each the subject of active lines of research.

Example 5.3 (Better Objectives and Unrolling Strategies). One path to consider is that the objective of the sparse rate reduction is not the best for causal sequence data. For example, we may want to ensure that the features have some kind of temporal relationships or structure, e.g., being generated by a linear dynamical system with Gaussian (mixture) initialization and Gaussian step noise. There is a recent but explosively popular line of work on efficiently and explicitly modeling dynamics in the features using so-called *state-space models* (SSMs, named such in contravention of the classical notion of "state space models" discussed in Chapter 1) [GD24; YKH24; YWZ+24]. Such models are empirically promising because they avoid the need for all-pairs attention, dodging the computational and memory complexity of the MSSA operator and its conventional cousin, the multi-head attention. Despite this efficiency gain, they are highly performant and have even been used in some large-scale language models [TZL+25]. By designing more sophisticated objectives and unrolling strategies, we may be able to build a "white-box" SSM architecture, or SSM-transformer hybrid, and thusly close the gap on causal tasks. ■

Example 5.4 (Inference-Time Computation Strategies). Another path to improve causal models is to use more so-called *inference-time compute* strategies: during inference (such as sampling from an LLM) find a way to spend more computational resources in order to obtain a better result. If we are able to do this, we can solve harder problems by allocating more compute to the task at hand.

One naive approach to this is to loop the model: feed the final features into the initial sequence-to-sequence layers and enable it to go through more rounds of iterative denoising. This methodology is known in the literature as "looped transformers" (for the case in which the backbone model is an empirically-designed transformers); this approach has been shown to have benefits at some particularly difficult tasks [GRS+23; SDL+25; YLN+23]. However, from our perspective it is more reasonable to loop the iterate through the *same layer*

multiple times, taking multiple local denoising or compression steps against the *same model* at each layer before going to the next.

Still another potential methodology relies on our understanding of the low-dimensional structure in the features and model parameters. For example, since the $\boldsymbol{U}_{[K]}$ matrices are supposed to be supports for the subspaces spanned by the features, we could update them at inference-time using an online algorithm such as online (G)PCA. Similar steps may be able to adapt the dictionary at inference time. This would allow us to improve and specialize our local denoising or compression operators at each layer for the features, and thus potentially yielding better performance.

There have been many empirical attempts to improve inference-time compute methods. It is still a very active area of ongoing research. ■

5.4 Summary and Notes

The materials presented in this chapter are based on a series of recent works on this topic, including [CYY+22; WLP+24; WLY+25; WDL+25; YBP+23]. These contributions encompass both theoretical advances and practical methodologies for constructing interpretable deep networks through unrolled optimization. Many of the key results and proofs discussed in this chapter are derived directly from, or inspired by, these foundational works.

The idea of unrolling an optimization algorithm to construct a neural network traces back to the seminal work [GL10]. In this work, the authors demonstrated that sparse coding algorithms—such as the Iterative Shrinkage-Thresholding Algorithm (ISTA)—can be unrolled to form multilayer perceptrons (MLPs), effectively bridging iterative optimization and neural network design. Notably, [MLE19] demonstrated that such unrolled networks are more interpretable, efficient, and effective compared to generic networks.

In this chapter, we build on this perspective to develop principled, white-box deep network architectures by unrolling optimization algorithms that are designed to minimize well-motivated objectives—such as the (sparse) rate reduction objective introduced earlier. This approach not only clarifies the role of each layer in the network but also offers theoretical grounding for architectural choices, moving beyond empirical trial-and-error toward interpretable and goal-driven design. In the following, we compare conventional DNNs, which are typically constructed through empirical design and heuristic tuning, with our mathematically grounded ReduNet architectures:

	Conventional DNNs	ReduNets
Objectives	input/output fitting	information gain
Deep architectures	trial & error	iterative optimization
Layer operators	empirical	projected gradient
Shift invariance	CNNs+augmentation	invariant ReduNets
Initializations	random/pre-design	forward unrolled
Training/fine-tuning	back prop	forward/back prop
Interpretability	black box	white box
Representations	hidden/latent	incoherent subspaces

5.5 Exercises and Extensions

Exercise 5.1. Let $\boldsymbol{Z} = [\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_K] \in \mathbb{R}^{d \times N}$ with $\boldsymbol{Z}_k \in \mathbb{R}^{d \times N_k}$ for each $k \in [K]$. For some $\alpha > 0$, let

$$R(\boldsymbol{Z}) = \log\det\left(\boldsymbol{I} + \alpha \boldsymbol{Z}\boldsymbol{Z}^\top\right).$$

1. Given any direction $\boldsymbol{D} \in \mathbb{R}^{d \times m}$, please show that $\nabla R(\boldsymbol{Z}) = \alpha \boldsymbol{X}^{-1}\boldsymbol{Z}$ and

$$\begin{aligned}&\nabla^2 R(\boldsymbol{Z})[\boldsymbol{D}, \boldsymbol{D}]\\ &= \alpha \mathrm{Tr}\left(\boldsymbol{X}^{-1}\boldsymbol{D}\boldsymbol{D}^\top\right) - \frac{\alpha^2}{2}\mathrm{Tr}\left(\boldsymbol{X}^{-1}\left(\boldsymbol{Z}\boldsymbol{D}^\top + \boldsymbol{D}\boldsymbol{Z}^\top\right)\boldsymbol{X}^{-1}\left(\boldsymbol{Z}\boldsymbol{D}^\top + \boldsymbol{D}\boldsymbol{Z}^\top\right)\right),\end{aligned}$$

   where $\boldsymbol{X} \doteq \boldsymbol{I} + \alpha \boldsymbol{Z}\boldsymbol{Z}^\top$. *Hint:* Note that

$$\nabla^2 R(\boldsymbol{Z})[\boldsymbol{D}, \boldsymbol{D}] \doteq \left\langle \lim_{t \to 0} \frac{\nabla R(\boldsymbol{Z} + t\boldsymbol{D}) - \nabla R(\boldsymbol{Z})}{t}, \boldsymbol{D} \right\rangle.$$

2. Please show that

$$R(\boldsymbol{Z}) \leq \sum_{k=1}^{K} \log\det\left(\boldsymbol{I} + \alpha \boldsymbol{Z}_k \boldsymbol{Z}_k^\top\right),$$

   where the equality holds if and only if $\boldsymbol{Z}_k^\top \boldsymbol{Z}_l = \boldsymbol{0}$ for all $k \neq l \in [K]$.

3. Given some $\alpha > 0$, let $\alpha_k = N\alpha/N_k$ for each $k \in [K]$. Please derive the closed-form for the first-order critical point of the following function:

$$f(\boldsymbol{Z}_k) = \frac{1}{2}\log\det\left(\boldsymbol{I} + \alpha \boldsymbol{Z}_k \boldsymbol{Z}_k^\top\right) - \frac{N_k}{2N}\log\det\left(\boldsymbol{I} + \alpha_k \boldsymbol{Z}_k \boldsymbol{Z}_k^\top\right) - \frac{\lambda}{2}\|\boldsymbol{Z}_k\|_F^2.$$

   Hint: Let $r_k = \mathrm{rank}(\boldsymbol{Z}_k)$. Consider the following singular value decomposition of $\boldsymbol{Z}_k$:

$$\boldsymbol{Z}_k = \boldsymbol{P}_k \boldsymbol{\Sigma}_k \boldsymbol{Q}_k^\top = \begin{bmatrix}\boldsymbol{P}_{k,1} & \boldsymbol{P}_{k,2}\end{bmatrix} \begin{bmatrix}\tilde{\boldsymbol{\Sigma}}_k & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0}\end{bmatrix} \begin{bmatrix}\boldsymbol{Q}_{k,1}^\top \\ \boldsymbol{Q}_{k,2}^\top\end{bmatrix}, \tag{5.5.1}$$

where $\boldsymbol{P}_k \in \mathcal{O}^d$ with $\boldsymbol{P}_{k,1} \in \mathbb{R}^{d\times r_k}$ and $\boldsymbol{P}_{k,2} \in \mathbb{R}^{d\times(d-r_k)}$, $\boldsymbol{\Sigma}_k \in \mathbb{R}^{d\times N_k}$ with $\tilde{\boldsymbol{\Sigma}}_k \in \mathbb{R}^{r_k\times r_k}$ being a diagonal matrix, and $\boldsymbol{Q}_k \in \mathcal{O}^{N_k}$ with $\boldsymbol{Q}_{k,1} \in \mathbb{R}^{N_k\times r_k}$ and $\boldsymbol{P}_{k,2} \in \mathbb{R}^{N_k\times(N_k-r_k)}$.

Exercise 5.2 (Neumann series for matrix inverse). Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$. If $\|\boldsymbol{A}\| < 1$, please show

$$(\boldsymbol{I} - \boldsymbol{A})^{-1} = \sum_{k=0}^{\infty} \boldsymbol{A}^k. \tag{5.5.2}$$

Hint: The proof consists of two steps.

(i) **Step 1**: Please show that the infinite series $\sum_{k=0}^{\infty} \boldsymbol{A}^k$ converges when $\|\boldsymbol{A}\| < 1$ using $\|\boldsymbol{A}^k\| \le \|\boldsymbol{A}\|^k$.

(ii) **Step 2**: Compute the matrix product $(\boldsymbol{I} - \boldsymbol{A})\sum_{k=0}^{\infty} \boldsymbol{A}^k$.

Exercise 5.3. In this exercise we will consider a coding rate measure $R_\epsilon^{c,\mu}$ which uses the means of the distributions, i.e., a coding rate for tokens drawn (non-i.i.d.) from the low-rank Gaussian mixture model

$$\boldsymbol{z}_i \sim \frac{1}{K}\sum_{k=1}^{K} \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{U}_k\boldsymbol{U}_k^\top). \tag{5.5.3}$$

1. Please argue that a suitable coding rate for $\boldsymbol{Z}$ under this model is

$$R_\epsilon^{c,\mu}(\boldsymbol{Z}) = \sum_{k=1}^{K} R_\epsilon(\boldsymbol{U}_k^\top(\boldsymbol{Z} - \boldsymbol{\mu}_k\boldsymbol{1}^\top)). \tag{5.5.4}$$

2. Similarly to how we derived MSSA, derive a new attention-like operator from this coding rate. What are the key differences?

Exercise 5.4. Please compute the gradients in (5.3.9) and (5.3.10).

Exercise 5.5. Please show Corollary 5.1 when $Kp \le d$.

Exercise 5.6. Derive a causal version of the TSSA operator.

Algorithm 5.1 Training algorithm for ReduNet

Input: $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N] \in \mathbb{R}^{D \times N}$, $\boldsymbol{\Pi} = \{\boldsymbol{\Pi}_k\}_{k=1}^K$, $\epsilon > 0$, λ, and a learning rate η.

Output: The learned parameters $\{\boldsymbol{E}^\ell\}_{\ell=1}^L, \{\{\boldsymbol{C}_k^\ell\}_{k=1}^K\}_{\ell=1}^L, \{\gamma_k\}_{k=1}^k$

procedure ReduNetTraining($\boldsymbol{X}, \boldsymbol{\Pi}, \epsilon, \lambda, \eta$)

`# Define constants`

$\alpha \leftarrow d/(N\epsilon^2)$

for $k \in \{1, \ldots, K\}$ **do**

$\alpha_k \leftarrow D/(\mathrm{tr}(\boldsymbol{\Pi}_k)\epsilon^2)$

$\gamma_k \leftarrow \mathrm{tr}(\boldsymbol{\Pi}_k)/D$

end for

`# ReduNet layer-by-layer iteration`

$\boldsymbol{Z}^1 = \left[\boldsymbol{z}_1^1, \ldots, \boldsymbol{z}_N^1\right] \leftarrow \boldsymbol{X}$ ▷ Initialize the ReduNet per-layer iteration

for $\ell \in \{1, \ldots, L\}$ **do**

`# Step 1: Compute network parameters` $\boldsymbol{E}^\ell, \{\boldsymbol{C}_k^\ell\}_{k=1}^K$

$\boldsymbol{E}^\ell \leftarrow \alpha\left(\boldsymbol{I} + \alpha \boldsymbol{Z}^\ell (\boldsymbol{Z}^\ell)^\top\right)^{-1} \in \mathbb{R}^{d \times d}$

for $k \in \{1, \ldots, K\}$ **do**

$\boldsymbol{C}_k^\ell \leftarrow \alpha_k\left(\boldsymbol{I} + \alpha_k \boldsymbol{Z}^\ell \boldsymbol{\Pi}_k (\boldsymbol{Z}^\ell)^\top\right)^{-1} \in \mathbb{R}^{d \times d}$

end for

`# Step 2: Update features` $\boldsymbol{Z}^\ell$

for $i \in \{1, \ldots, N\}$ **do**

`# Compute soft assignments` $\hat{\boldsymbol{\pi}}(\boldsymbol{z}_i^\ell)$

$\hat{\boldsymbol{\pi}}(\boldsymbol{z}_i^\ell) \leftarrow \mathrm{softmax}(-\lambda[\|\boldsymbol{C}_1^\ell \boldsymbol{z}_i^\ell\|_2, \ldots, \|\boldsymbol{C}_K^\ell \boldsymbol{z}_i^\ell\|_2]) \in [0,1]^K$

`# Update features` $\boldsymbol{z}_i^{\ell+1}$ `from` $\boldsymbol{z}_i^\ell$

$$\boldsymbol{z}_i^{\ell+1} \leftarrow \mathcal{P}_{\mathbb{S}^{d-1}}\left(\boldsymbol{z}_i^\ell + \eta\left(\boldsymbol{E}^\ell \boldsymbol{z}_i^\ell - \sum_{k=1}^K \gamma_k \hat{\pi}_k(\boldsymbol{z}_i^\ell) \boldsymbol{C}_k^\ell \boldsymbol{z}_i^\ell\right)\right) \in \mathbb{R}^d$$

end for

end for

`# Return all network parameters.`

return $\{\boldsymbol{E}^\ell\}_{\ell=1}^L, \{\{\boldsymbol{C}_k^\ell\}_{k=1}^K\}_{\ell=1}^L, \{\gamma_k\}_{k=1}^K$

end procedure

Algorithm 5.2 Evaluation algorithm for ReduNet

Input: Input $\boldsymbol{x} \in \mathbb{R}^D$, network parameters $\{\boldsymbol{E}^\ell\}_{\ell=1}^L, \{\{\boldsymbol{C}_k^\ell\}_{k=1}^K\}_{\ell=1}^L, \{\gamma_k\}_{k=1}^K$, learning rate λ

Output: feature $\boldsymbol{z}^{L+1}$

procedure ReduNetEvaluation($\boldsymbol{x}$)

    # Initialize the ReduNet per-layer iteration

    $\boldsymbol{z}^1 \leftarrow \boldsymbol{x} \in \mathbb{R}^D$

    for $\ell \in \{1, \ldots, L\}$ **do**

        # Compute soft assignments $\hat{\boldsymbol{\pi}}(\boldsymbol{z}^\ell)$

        $\hat{\boldsymbol{\pi}}(\boldsymbol{z}^\ell) \leftarrow \mathrm{softmax}(-\lambda \left[\|\boldsymbol{C}_1^\ell \boldsymbol{z}^\ell\|_2, \ldots, \|\boldsymbol{C}_K^\ell \boldsymbol{z}^\ell\|_2\right]) \in [0,1]^K$

        # Update feature $\boldsymbol{z}^{\ell+1}$ using $\boldsymbol{z}^\ell$

        $\boldsymbol{z}^{\ell+1} \leftarrow \mathcal{P}_{\mathbb{S}^{d-1}}\left(\boldsymbol{z}^\ell + \eta\left(\boldsymbol{E}^\ell \boldsymbol{z}^\ell - \sum_{k=1}^K \gamma_k \hat{\pi}_k(\boldsymbol{z}^\ell) \boldsymbol{C}_k^\ell \boldsymbol{z}^\ell\right)\right) \in \mathbb{R}^d$

    end for

    # Return the output features

    return $\boldsymbol{z}^{L+1}$

end procedure