Feature Alignment As A Generative Process

Tiago Souza Farias (tiago939@gmail.com)
Universidade Federal de Santa Maria
Jonas Maziero
Universidade Federal de Santa Maria

Research Article

Keywords:

Posted Date: February 23rd, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1338863/v1

Feature alignment as a generative process

Tiago de Souza Farias* and Jonas Maziero

Departamento de Física, Centro de Ciências Naturais e Exatas, Universidade Federal de Santa Maria, Avenida Roraima 1000, Santa Maria, Rio Grande do Sul, 97105-900, Brazil

ABSTRACT

We introduce feature alignment, a technique for obtaining approximate reversibility in artificial neural networks. By means of feature extraction, we can train a neural network to learn an estimated map for its reverse process from outputs to inputs. Combined with variational autoencoders, we can generate new samples from the same statistics as the training data. Improvements of the results are obtained by using concepts from generative adversarial networks. Finally, we show that the technique can be modified for training neural networks locally, saving computational memory resources. Applying these techniques, we report results for three vision generative tasks: MNIST, CIFAR-10, and celebA.

1 Introduction

Feature visualization[1] is a set of techniques for neural networks aiming to find inputs that maximize the activation of one or more selected neurons from the same network. Usually, feature visualization is used as a method for model interpretability, where one seeks to understand a neural network by analyzing how much each neuron contributes to a neural network by perceiving the images generated by these techniques. The process of obtaining these inputs is, in a sense, an attempt towards reversing a neural network. Since a neural network is composed by functions that map inputs to outputs, the visual representation of a feature is the input we would have given a target activation for a group of posterior selected neurons.

The reversibility of neural networks relates to how well one can reverse the map from the activation of target neurons to the input neurons[2]. Usually, neural networks are not reversible, due mainly to three reasons: 1) the presence of non-reversible activation functions (e.g. ReLU[3]), such that in general one cannot directly recover an input x given the value $f(x)$. 2) Non-orthogonal weights, since there are no constrains or incentive for the matrix representation to converge to have this property. 3) Lack of one-to-one relationships, due to the reduction of dimension as the information is passed through each layer of a network. One advantage of reversible neural networks, besides the reversing mapping, is their memory efficiency: while non-invertible networks need to store all the activations for the backward pass during training, reversible networks allow the storage of fewer activations to update the trainable parameters.

Reversibility also constrains the number of possible models, as many possible parameters configurations model the data. For example, if one considers an analytical function that one wishes to approximate by a sufficiently parameterized neural network, with the pair of data $\{x, f(x)\}$, several local minima estimate the function $x \to f(x)$, each one obtained by a different random initialization of the neural network parameters (assuming optimal convergence). By restricting the reversibility $f(x) \to x$, we can reduce the number of optimal points toward which a neural network can converge. While local optima are not a problem for neural networks, since many of them converge to similar losses, they lack interpretability since one can not recover the inputs given a specific output.

Memory is often a bottleneck for neural networks. The backpropagation algorithm[4,5], widely used for training in modern deep learning, requires the storage of the activation of all neurons of a network to update its parameters. Local training rules[6] allow a more efficient memory optimization of neural networks. By constraining the trainable parameters, such as the weights, to be updated only by local variables (the information contained in the neurons that share the same parameter), we can reduce the memory requirements to load a model in hardware such as CPUs and GPUs. This constraint can save memory resources and has many potential applications, from low-memory devices[7,8] to train large batch sizes[9,10], and, even further, to train very large neural networks[11].

In this article, we show that feature visualization can be used for approximate reversibility of neural networks on the training data. This approximation relies on doing gradient descent on the input space and training a network to estimate the input given an output. To show the feasibility of the proposed technique, we use the concept of generative networks to generate samples statistically similar to the training data by making use of approximate reversibility. We also adapt the technique for local training, showing that is possible to reverse an encoder by mapping the output latent vector back to the images of a dataset.

2 Related work

Several works have been done in the area of feature extraction, especially applied for model interpretability and explainability[12–14]. These techniques, used for extracting features, usually consist in activation maximization[15], where a group of neurons, which can involve from a single neuron up to an entire layer (or channel for convolutions), is selected to extract the feature by maximizing its activation. Many of these techniques of feature extraction consist in studying features in already pre-trained classifiers[16]. Other techniques consist in searching for features in the latent space[17]. Feature extraction can also be utilized for understanding which parts of an input contribute the most for the target activations[18–20].

In a generative process, we want to produce new examples with the same statistical distribution as the training data. There are several different techniques to model the data for a generation. Among these techniques, autoencoder based networks, generative adversarial networks, and normalizing flows are very popular. Autoencoders (AE), while not generative networks, they constitute of building blocks for other generative networks and offer insights about mapping the input to other representations. Autoencoder consist of two networks: an encoder that projects the inputs into a vector, usually with a smaller dimension, and a decoder that reconstructs the input from this vector. The compressed vector has a high-level representation of the model, in which each neuron contributes to properties beyond the data level at the input layer[21]. Autoencoders are commonly trained in an unsupervised fashion, nevertheless, some variants include labeled information to further increase training for a specific objective. Variational autoencoders (VAE)[22–24] gives autoencoders generative capability by projecting the data into a probabilistic latent vector, thus we can generate data statiscally similar to the training data by sampling random latent vectors and projecting them to a decoder network. Generative adversarial networks (GANs)[25–27] are another example of a generative method. By having two competing networks, a generative network which takes a random low-dimensional input and outputs an image, and a discriminator network that compares the images from the training dataset and the sampled ones from the generator. The competition arises by training the generator to fool the discriminator by generating images as closest to the training dataset as possible. Normalizing flows[28–30] is another generative paradigm that generate images by transforming a simple distribution to a more complex one by a series of reversible transformations.

There have been works combining autoencoders with GANs[31]. The work done in Refs. 32, 33 is related to ours. They synthesized new images with the same statistics as the training data by imputing features to a generator network. The main difference is that, in these previous articles, the features are obtained with a pretrained network.

Most works on reversibility consist in architectural changes of neural networks[34–38]. These changes guarantee a one-to-one relationship between inputs and outputs. BiGAN[39] constructs a generative network and a reverse network that inputs images back to noise, which can be used to obtain a latent representation of a dataset directly.

Local learning rules have been explored since Donald Hebb proposed a simple model for learning in the brain[40]. The main advantage of this kind of learning algorithm is requiring lower memory resources. Some works are biologically inspired[41,42], while others focus solely on efficiency[43,44]. There is a growing body of work discussing whether the brain does backpropagation[45,46], with some approximations for training artificial neural networks[47–49].

Another approach for saving memory resources is gradient checkpoint[50–53], where memory is traded with computation time by re-evaluating neurons when they are needed for backpropagation instead of storing their activations all at once. While this technique decreases the amount of memory necessary to train a neural network, it requires many forward propagation calculations on the network, depending on its size, which can increase time consumption, while local learning rules, as opposite, require only one forward propagation to update the parameters.

3 Methods

Here we discuss the technique of feature alignment. It consists of two steps: first we do a gradient descent on a random input with respect to a loss function that measures the distance between the output of the random input and the true input. Second, we train the network on a new loss function that measures the distance between the inputs and the gradient we did on it. By doing this, the network learns the inverse map from the outputs back to its corresponding inputs, thus recovering the information about what led to its activation.

3.1 Feature alignment

Let $E(\mathbf{x};\theta)$ be a neural network with inputs $\mathbf{x}$ and trainable parameters θ. The output is a latent vector $\mathbf{z}_x$ with an arbitrary number of neurons. The feature alignment encoder consists of a neural network with parameters θ and an arbitrary number of latent variables as the output. From a dataset $\mathbf{x} \in \mathbf{X}$, $\mathbf{z}_x = E(\mathbf{x};\theta)$ is the output from an input $\mathbf{x}$. With the same network, $z_r = E(\mathbf{r};\theta)$ is the output from a random input $\mathbf{r}$, choosen from some propability distribution, with the same dimension as the input data.

The feature $\hat{\mathbf{r}}$ of $\mathbf{z}(\mathbf{x})$ is obtained by minimizing a distance function $\mathscr{L}(\mathbf{z}_x, \mathbf{z}_r)$ with respect to the random inputs $\mathbf{r}$. We choose a gradient flow for minimizing this distance, since it can evolve the random input continually, as follows in the equation 1.

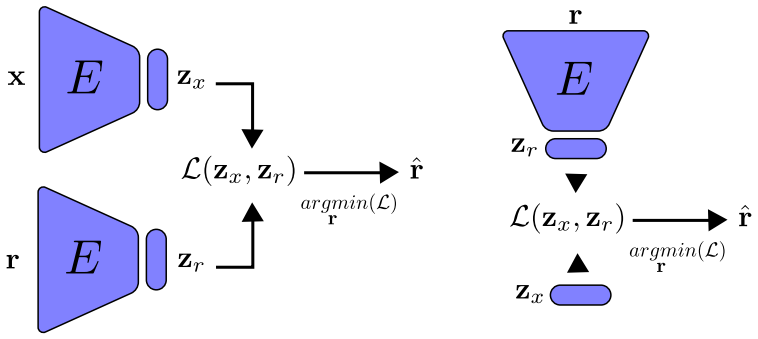


Figure 1. In feature alignment, we have two encoders that share the same parameters. Left: at training time, the network is trained to approximate $\hat{\mathbf{r}} \approx \mathbf{x}$. Right: during inference, we sample $\mathbf{z}_x$ from either the data or a known distribution to obtain the features.

$$\frac{\partial r}{\partial t} = -\frac{\partial \mathscr{L}}{\partial r}. \tag{1}$$

Since $\mathbf{z}(\mathbf{x})$ is fixed, $\mathbf{r}$ will evolve such as the function of the random variables will approximate $\mathbf{z}(\mathbf{x})$ as much as possible (See Appendix A.1). We want to solve equation (1) as efficiently as possible in time and memory. By discretizing the gradient flow, we obtain an approximation for the feature in equation 2.

$$r^t = r^{t-1} - \tau \frac{\partial \mathscr{L}}{\partial r}, \tag{2}$$

with τ being a hyperparameter that weights the contribution of the gradient. Equation 2 is similar to activation maximization, except that we are minimizing for the neurons to have a target activation. These updates are done in T time steps. Properly optimized, the solution to equation (2) converges to a feature by approximating the inverse of the weights (see Appendix A.1). So, by optimizing the parameters of the network, the weight matrix between layers will have the orthogonal property $\mathbf{w}^T\mathbf{w} = \mathbf{I}$, which implies in approximated reversibility (see Appendix A.2).

After we extract the feature $\hat{\mathbf{r}}$, we measure how similar it is to the inputs $\mathbf{x}$ by a new loss function $\mathscr{L}(\mathbf{x}, \hat{\mathbf{r}})$. This second loss function is used for training the encoder by optimizing its parameters. As the neural network is trained, the encoder learns, not only to map the inputs to the latent variables, but also the reconstruction of the inputs from the latent vector. Following training, we can reconstruct the inputs by knowing only the latent vector. Figure 1 and Algorithm 1 summarize the feature alignment technique. Notice that we initialize $\mathbf{r}$ with the same shape as $\mathbf{x}$.

Algorithm 1 Training with feature alignment

$\mathbf{z_x} = E(\mathbf{x}; \theta)$
initialize $\mathbf{r} = \mathbf{x}.shape$
$t = 0$
while $t < T$ **do**
  $\mathbf{z_r} = E(\mathbf{r}; \theta)$
  $\mathscr{L} = ||\mathbf{z_x} - \mathbf{z_r}||_2^2$
  $r = r - \tau \frac{\partial \mathscr{L}}{\partial r}$
  $t = t + 1$
end while
$\hat{\mathbf{r}} = \mathbf{r}$
$\mathscr{C} = ||\mathbf{x} - \hat{\mathbf{r}}||_2^2$
update θ by optimizing $\mathscr{C}$

3.2 Variational autoencoders with feature alignment (VFA)

In the context of generative processes, one issue with the encoder, and autoencoders in general, is its inability to generate new samples with the same statistical distribution as the training data. The latent variables from the data are associated with a distribution of variables that might be too complicated for effective sampling. To solve this problem, we use a variational

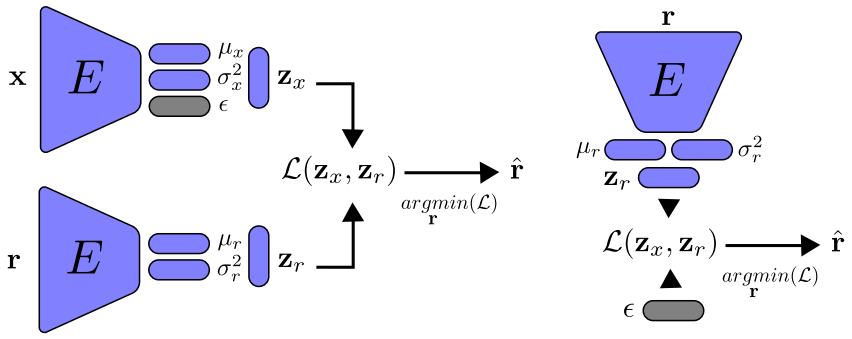


Figure 2. Variational autoencoder with feature alignment. Left: training the encoder to reconstruct the inputs $\mathbf{x}$. Right: we sample a random normal vector ε to generate new data.

autoencoder (VAE) formulation, except by removing the decoder network, by doing this, the inverse of the encoder becomes its own decoder. In the VAE, the output of the encoder is coupled with two layers that return the mean value μ_x and variance σ_x^2 of the data. We constrain the latent vector to have a distribution that is easy to sample (usually a Gaussian distribution), by comparing two probability distributions via a metric such as the Kullback-Leibler divergence. So, the cost function in equation 3 is used to train the feature encoder with a constraint to the output latent variables from a known random probability distribution $p(\mathbf{z})$, from which we can easily sample. The constant β is a hyper-parameter that regularizes the latent vector for better disentanglement representation of the data[54–56].

$$\mathscr{L} = ||\mathbf{x} - \hat{\mathbf{r}}||_2^2 - \beta \mathscr{D}_{KL}(q_\theta(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) \tag{3}$$

We choose the distribution $p(\mathbf{z})$ according to the principle of maximum entropy: since the latent variables are in the range $(-\infty, +\infty)$, the Gaussian distribution is best suited in this case. We then constrain each latent variable to come from a Gaussian distribution with zero mean value and variance equal to one. Just like the VAE, we cannot train the encoder by sampling directly from the mean and variance of the latent vector. Instead, we use the re-parametrization trick: we sample a random vector ε from a normal distribution, the latent vector is then: $\mathbf{z_x} = \mu_x + \varepsilon \odot \sigma_x^2$, with $\odot$ the element-wise product. Note that we do not have a random normal vector for $\mathbf{z_r}$, since its purpose is solely to reconstruct $\mathbf{x}$. Figure 2 summarizes training a VAE with feature alignment.

3.3 Improving the quality of the features

As will be shown in the results section, the images extracted using the feature alignment trained with VAE are blurry, due to the variational autoencoder nature[57]. To improve the quality of the generated images, we couple a generator network G and a discriminator network D to the images generated by the technique. In this way, the feature acts as a second latent vector, with a more complex distribution sampled from the simpler ones. This generator network is similar to the refiner network presented in Ref. 34, which takes an image as input and outputs an improved version of it.

The generator takes the feature as input and produces a new output which is compared to the input $\mathbf{x}$. The discriminator is trained as a generative adversarial network, evaluating the probability that $G(\hat{\mathbf{r}})$ is real or fake. The generator is updated by receiving gradients from the discriminator. For more stable training, we use the least square loss for the discriminator[58]. Alternatively, we can use the Wasserstein GAN formulation[59], which replaces the discriminator with a critic network that measures the score of the "*realness*" of an image.

To reduce possible effects due to the posterior collapse problem in VAEs[60–62] and to balance with the reconstruction loss, we propose a random schedule for the β variable. For each example during training, we sample β from a uniform distribution $\mathscr{U}(0,1)$.

While the reconstruction of images is optimized usually with pixel-level loss, we can also consider the high-level properties of the data. This kind of measure, called perceptual loss[63], compares the output of the reconstruction with the original image at high-level neurons (presented near the end of the network). Here we consider the perceptual loss using the mean and variance layers from the encoder network, enforcing the reconstruction to have the same statistical properties of the original input.

The final losses, for the encoder, generator, and discriminator are shown in equations 4, 5 and 6 respectively:

$$\mathscr{L}_E = ||\mathbf{x} - \hat{\mathbf{r}}||_2^2 + \beta \mathscr{D}_{KL}(q_\theta(\mathbf{z}|\mathbf{x})||p(\mathbf{z})), \tag{4}$$

$$\mathscr{L}_G = ||1 - D(G(\hat{\mathbf{r}}))||_2^2 + \lambda(||\mu(\mathbf{x}) - \mu(G(\hat{\mathbf{r}}))||_2^2 + ||\sigma^2(\mathbf{x}) - \sigma^2(G(\hat{\mathbf{r}}))||_2^2), \tag{5}$$

$$\mathscr{L}_D = ||1 - D(\mathbf{x})||_2^2 + ||-1 - D(G(\hat{\mathbf{r}}))||_2^2. \tag{6}$$

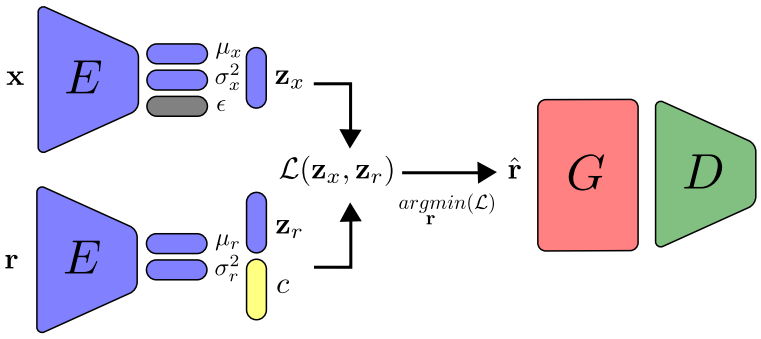


Figure 3. Training in the VFA setting, with the addition of generator and discriminator networks.

with λ a hyperparameter that weights the perceptual loss contribution.

If a training data has additional information, such as labels, we can train a neural network jointly with supervised training for specific related tasks such as conditional generation, where we want samples that correspond to a desired class. We can condition the latent variables to have different distributions from different classes. The output of the network is trained from the Gaussian distribution $\mathbf{z} \sim Q(z_i, c_i)$, with c_i being a one-hot vector containing the class information. Similar to the work done in Ref. 64, we couple a linear classification layer on top of the network, parallel to the mean and variance layers. Since the class layer is linear, at inference time we can choose a greater value for the one-hot vector, which puts emphasis on the chosen class. The improvements made on the variational autoencoders with feature alignment discussed in this section are illustrated in figure 3.

3.4 Local feature alignment

The rules for feature alignment were presented as a global rule: the auxiliary loss is defined with the output layer and the loss is defined with the input layer, so we have full communication between all layers. However, we can reformulate this rule with local losses, similar to target propagation rules[65–67]: the auxiliary loss and loss are defined as the interaction between two connected layers only (or even individual neurons), as follows: for each layer from the first to the last, we activate it from its inputs and store a second activation from a random input with the same dimension, we then extract the feature by optimizing the random input with an auxiliary loss between activation of the random output and the true input. Finally, the parameters of the chosen layer are updated by optimizing the loss between the reconstruction and true input. This technique of local training is summarized in Algorithm 2 and illustrated in figure 4.

Algorithm 2 Training with local feature alignment

for l=0, L **do** ▷ for each layer
  $\mathbf{z}_x^l = E(\mathbf{x}; \theta_l)$
  initialize $\mathbf{r} = \mathbf{x}.shape$
  $t = 0$
  while $t < T$ **do**
    $\mathbf{z}_r^l = E(\mathbf{r}; \theta_l)$
    $\mathscr{L}_l = ||\mathbf{z}_x^l - \mathbf{z}_r^l||_2^2$
    $r = r - \tau \frac{\partial \mathscr{L}_l}{\partial r}$
    $t = t + 1$
  end while
  $\hat{\mathbf{r}} = \mathbf{r}$
  $\mathscr{C}_l = ||\mathbf{x} - \hat{\mathbf{r}}||^2$
  update θ_l by optimizing $\mathscr{C}_l$
  $\mathbf{x} = \mathbf{z}_x^l.detach$
end for

The neural network becomes a predictive machine, where each layer trains its parameters by trying to predict the inputs by knowing the outputs. The constraint of the local learning has a more pronounced effect on the non-linearity of a neural network trained in this way. While backpropagation can adjust all the parameters of a network so that the feature reconstructs the input, local rules can only rely on very strict information content available. Non-reversible functions, such as the

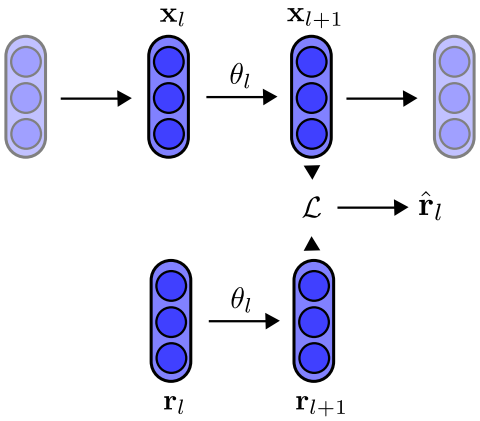


Figure 4. Illustration of the local training rule.

ReLU function, propagate loss of information, which will lead to reconstructions with low fidelity. Thus, a non-linear function must be carefully chosen in order to preserve as much information as possible. The function *inverse hyperbolic sine* ($\text{arcsinh}(x) = \ln(x + \sqrt{1+x^2})$), is similar to the *hyperbolic tangent* near zero and logarithmic at large (absolute) values. This function has the properties of being fully invertible, zero-centered mean, unbounded, continuously differentiable and its gradient does not vanish as fast as for tanh. These properties make arcsinh a good candidate function for local training.

At inference time, we need to propagate the information backward from the output to the input layer by layer just as one would do normally with non-local feature extraction. However, the non-linear function will play a major role here, it being reversible will be required to approximate reversibility. This is done by, after inputting the latent vector, applying the inverse of the non-linear function after each layer that utilizes the function. This process is summarized in the Algorithm 3.

Algorithm 3 Reconstruction with local feature alignment

sample $\mathbf{z}_x^L$
for l=L, 0 **do**
  initialize $\mathbf{r} = \mathbf{x}.shape$ with $\mathbf{x}$ as $\mathbf{z}_x^l = E(\mathbf{x}; \theta_l)$
  $t = 0$
  while $t < T$ **do**
    $\mathbf{z}_r^l = E(\mathbf{r}; \theta_l)$
    $\mathscr{L}_l = ||\mathbf{z}_x^l - \mathbf{z}_r^l||_2^2$
    $r = r - \tau \frac{\partial \mathscr{L}_l}{\partial r}$
    $t = t + 1$
  end while
  $\hat{\mathbf{r}} = \mathbf{r}$
  if layer = non-linear function f **then**
    $\hat{\mathbf{r}} = f^{-1}(\hat{\mathbf{r}})$
  end if
  $\mathbf{z}_x^l = \hat{\mathbf{r}}$
end for

4 Implementation details

The encoder network consists of a series of convolutional layers, similar to the AlexNet architecture[68], but with stride one and two for down-scaling, instead of maxpool, with LeakyReLU activation. The generator network has three convolutional layers. The discriminator network has the same architecture as the encoder, except for the last layer that outputs a single value for each example. Only the generator utilizes batch normalization after each convolution. All convolutions have kernel size $k = 3$. Details of the networks can be found in tables 2, 3, 4 for MNIST, CIFAR-10 and CelebA respectively.

We use the Adam optimizer[69] with learning rate $\eta = 0.00001$ and batch size 128. The parameters of the encoder and generator networks are initialized with orthogonal initialization[70,71]. We set the hyperparameter $\lambda = 0.01$ and sample β from a uniform distribution, with a different random value for each training example.

From Appendix A.2, we have that the loss become unstable when the weights $w^2 > 2$, so we restrict the weights to the range $-\sqrt{2} \leq w \leq \sqrt{2}$ by clamping then, as shown in equation 7.

$$w = \begin{cases} -\sqrt{2} & \text{if } w < -\sqrt{2}, \\ \sqrt{2} & \text{if } w > \sqrt{2}, \\ w & \text{otherwise.} \end{cases} \tag{7}$$

For the image datasets, the generator is composed of a few convolutional layers, with the same number of filters and kernel sizes.

We also report the results for the modified feature alignment for local training as a proof of concept by training an encoder for reconstruction from a latent vector.

For GAN, when used for reconstruction, we search the latent space that leads to most similar images by optimizing $\arg\min_z ||x - G(z)||_2^2$.

5 Results

We compare the results against traditional variational autoencoders and generative adversarial networks. The results show the reconstruction of the inputs with feature and generator. We also show random samples from the generator network. We evaluate the Fréchet Inception Distance (FID)[72,73] (equation 8) for each dataset as a measure of the quality of the samples. The FID score is calculated by extracting the activation of the global spatial pooling layer of a pre-trained Inception V3 model[74], for equally numbered images from the dataset (here we choose 10k images) and sampled from a generator model:

$$FID = ||\mu_1 - \mu_2||_2^2 + tr(\Sigma_1 + \Sigma_2 - 2\sqrt{\Sigma_1 \Sigma_2}) \tag{8}$$

with μ the mean of activations, Σ the covariance matrix and tr the trace function. The average results are shown in Table 1 for three different different initializations.

Method	MNIST	CelebA	CIFAR-10
VAE	39.84 ± 0.15	84.84 ± 0.10	163.59 ± 0.37
GAN	21.50 ± 2.79	32.85 ± 1.22	63.39 ± 0.62
$\hat{\mathbf{r}}$ **(ours)**	120.02 ± 1.11	143.51 ± 1.04	209.32 ± 1.96
$G(\hat{\mathbf{r}})$ **(ours)**	41.24 ± 2.71	132.18 ± 2.73	73.20 ± 2.67

Table 1. FID scores across three image datasets. Mean and standard deviation from three trials.

5.1 MNIST

The MNIST dataset[75] is a set of 60000 grayscale images with size 28×28 pixels that contain hand drawings of digits from zero to nine. Figure 5a shows the latent space trained on two neurons as outputs, we can see that the network tries to cluster the images to similarity, while figure 5b shows the reconstruction for a fixed latent vector but varying a trained classified output vector according to the labels of the dataset. Figure 6 shows the reconstruction of images by the features and with the generator applied to them, compared with traditional AE, VAE, and GAN. While the features tend to be noisy, the generator can sharpen the images to resemble better the original inputs. Figure 7 shows random samples from the generator with a class layer coupled to the encoder. The same image also shows the interpolation of the latent vector between four pairs of images to demonstrate the continuity of transitions on the latent space.

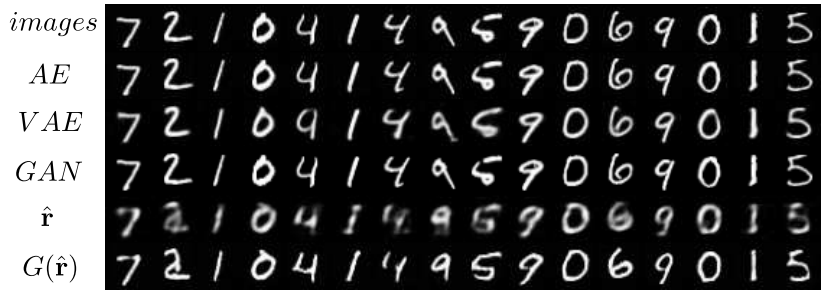
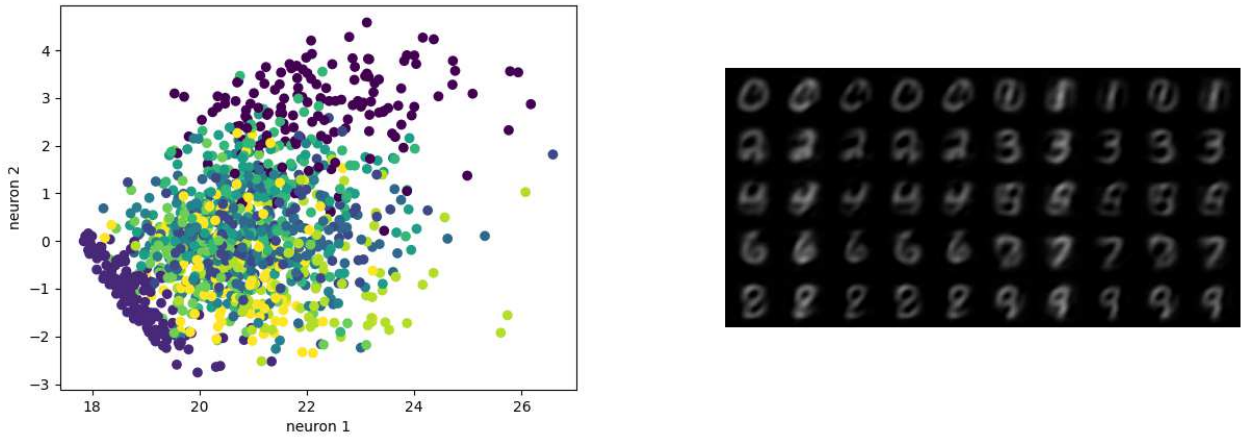


Figure 6. Reconstruction of images of the MNIST dataset from four different models.



(a) (b)

Figure 5. Representation of the latent space. (a) Latent space with two neurons, (b) images from features extracted by manipulating the classification layer.

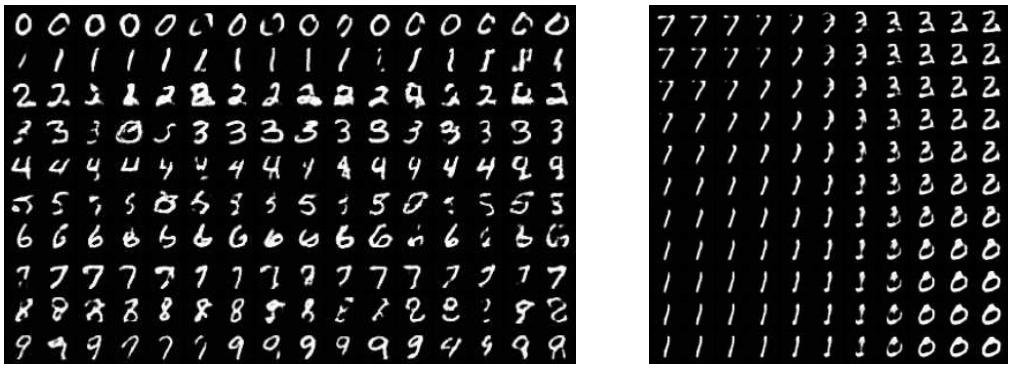


Figure 7. Left: random samples. Right: interpolation among four images reconstructed from the dataset.

5.2 CelebA

The CelebA dataset[76] is a set of 202600 images of celebrity faces. We resize the images to 64×64 pixels. Figures 8 and 9 show the reconstruction and sampling with interpolation between samples, respectively. Without the perceptual loss (with $\lambda = 1$), we noticed a failure on the convergence of the generator network.

Figure 8. Reconstruction of images of the CelebA dataset from four different models.

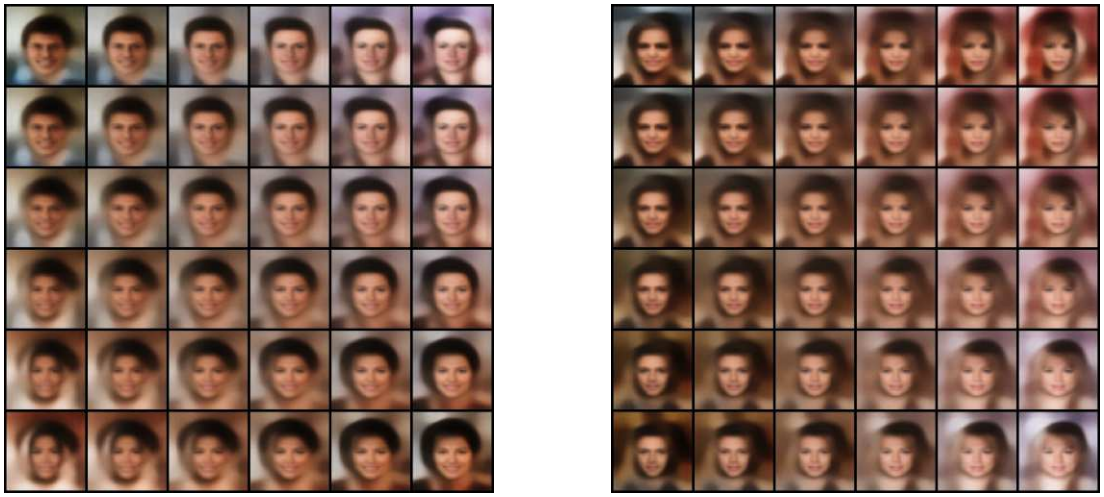


Figure 9. Two sets of interpolation among four random sampled images.

5.3 CIFAR-10

The CIFAR-10 dataset[77] contains 70000 natural images with size 32×32 pixels across 10 different classes. Figure 10 shows the reconstruction of images from the dataset. While the features do approximate the original inputs, the transformation of the generator tends to be more dissimilar due to its loss being dependent only on the adversarial contribution ($\lambda = 0$). Just as before, Figure 11 shows random samples and interpolation, which show a diversity of images, albeit less perceptual similar to the original dataset.

Figure 10. Reconstruction of images on CIFAR-10.

Figure 11. Left: random samples. Right: interpolation among four images reconstructed from the dataset.

It is important to note that we do not expect feature alignment to exceed the reconstruction and sample qualities of VAEs and GANs. The condition of reversibility constrains the optimization of a neural network and thus has to balance the reversibility

cost with other losses. Nevertheless, we compare the reconstruction L_2 loss with other networks to analyze how different each network is compared to the same metric. These results are shown in figure 12.

As expected, autoencoders have the lowest loss, given that it is optimized directly for reconstruction. The features should be compared to VAEs and the generator with GAN since they are optimized with similar losses. Here we can see that absence of perceptual loss on the generator network decreases the reconstruction (without optimizing the latent vector) ability.

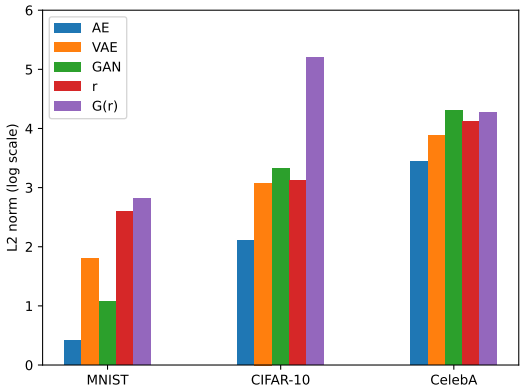


Figure 12. Comparison of the reconstruction L_2 loss for AE, VAE, GAN, r, and $G(r)$.

5.4 Local feature alignment

Here we show results from an encoder trained for reconstruction with the local feature alignment training. Figures 13, 14, and 15 show reconstruction pairs for the MNIST, CIFAR-10, and CelebA datasets respectively. We can see that local training can reconstruct images despite the layers not receiving information from the reconstruction loss of images, which can be explained with the same reason of non-local feature alignment: the weights form an orthogonal matrix which tries to reverse information between layers as much as possible, which is limited only by the network capacity, directly related to the number of neurons.

Figure 13. Local feature alignment on the MNIST dataset. Each pair of images contains the reconstruction and original, respectively.

Figure 14. Local feature alignment on the CIFAR-10 dataset. Each pair of images contains the reconstruction and original, respectively.

Figure 15. Local feature alignment on the CelebA dataset. Each pair of images contains the reconstruction and original, respectively.

6 Conclusions

We presented feature alignment, a technique that can approximately reverse neural networks. We trained an encoder to predict its input by optimizing the features to match the inputs. By coupling a probabilistic layer with the same formulation as the variational autoencoders, we can to generate new samples with the same statistical distribution of the training data. To improve the quality of the generated samples, that suffer from noise effects, we combined the generative adversarial network method by coupling a generator and a discriminator network to the features. We also showed that the technique can be modified to a local training rule (alternative to backpropagation), which offers the advantage of lowering memory resources for training and extracting gradients of neural networks.

Mathematical analysis on the convergence of the proposed technique shows that the weights converge to a pseudo-inverse matrix, which justifies the convergence of a network trained in this way to map its outputs back to its inputs. The restriction is the network's architecture itself, since the bottlenecks do not allow for a one-to-one relationship.

The results show that the features can approximately match the inputs. Despite not being able to improve on the sampling quality of other current generative techniques, reversibility can offer advantages when a mapping of the outputs back to its inputs is desired. The dataset CIFAR-10 is notoriously difficult because of the small image size and high variance, which led to samples with high FID measure. By coupling a classification layer on the encoder network, we can utilize label information for conditional generation of samples. Additionally, the results of local training suggests that we can train neural networks without feedback of global loss functions.

The proposed technique can be seen as a compromise between VAEs and GANs. The full architecture has crisper samples than VAEs and a latent vector that can be exploited. Feature extraction can be implemented across many different neural network architectures. Thus, it can potentially be improved to further increase the quality of the results.

Code availability

The code to reproduce the results is available on Github: `https://github.com/tiago939/feature_aligment`.

References

1. Olah, C., Mordvintsev, A. & Schubert, L. Feature visualization. *Distill* **2**, e7, DOI: 10.23915/distill.00007 (2017).
2. Gomez, A. N., Ren, M., Urtasun, R. & Grosse, R. B. The reversible residual network: Backpropagation without storing activations. *arXiv:1707.04585 [cs]* .
3. Nair, V. & Hinton, G. E. Rectified linear units improve restricted boltzmann machines. (2010).
4. Linnainmaa, S. Taylor expansion of the accumulated rounding error. *BIT Numer. Math.* **16**, 146–160, DOI: 10.1007/BF01931367 (1976).
5. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. Learning representations by back-propagating errors. *Nature* **323**, 533–536, DOI: 10.1038/323533a0 (1986).
6. Baldi, P. & Sadowski, P. A Theory of Local Learning, the Learning Channel, and the Optimality of Backpropagation. *Neural Networks* **83**, 51–74, DOI: 10.1016/j.neunet.2016.07.006 (2016).
7. Velichko, A. Neural Network for Low-Memory IoT Devices and MNIST Image Recognition Using Kernels Based on Logistic Map. *Electronics* **9**, 1432, DOI: 10.3390/electronics9091432 (2020).
8. Sohoni, N. S., Aberger, C. R., Leszczynski, M., Zhang, J. & Ré, C. Low-Memory Neural Network Training: A Technical Report. *arXiv:1904.10631 [cs, stat]* (2019).

9. Gao, F. & Zhong, H. Study on the Large Batch Size Training of Neural Networks Based on the Second Order Gradient. *arXiv:2012.08795 [cs]* (2020).

10. You, Y., Gitman, I. & Ginsburg, B. Large Batch Training of Convolutional Networks. *arXiv:1708.03888 [cs]* (2017).

11. Jing, K. & Xu, J. A Survey on Neural Network Language Models. *arXiv:1906.03591 [cs]* (2019).

12. Shahroudnejad, A. A Survey on Understanding, Visualizations, and Explanation of Deep Neural Networks. *arXiv:2102.01792 [cs]* (2021).

13. Fan, F., Xiong, J., Li, M. & Wang, G. On Interpretability of Artificial Neural Networks: A Survey. *arXiv:2001.02522 [cs, stat]* (2021).

14. Gilpin, L. H. *et al.* Explaining Explanations: An Overview of Interpretability of Machine Learning. *arXiv:1806.00069 [cs, stat]* (2019).

15. Mahendran, A. & Vedaldi, A. Visualizing Deep Convolutional Neural Networks Using Natural Pre-Images. *Int. J. Comput. Vis.* **120**, 233–255, DOI: 10.1007/s11263-016-0911-8 (2016).

16. Nguyen, A., Yosinski, J. & Clune, J. Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks. *arXiv:1602.03616 [cs]* (2016).

17. Shen, Y., Gu, J., Tang, X. & Zhou, B. Interpreting the latent space of gans for semantic face editing. *IEEE/CVF Conf. on Comput. Vis. Pattern Recognit. (CVPR), Seattle, WA, USA* 9240–9249, DOI: 10.1109/CVPR42600.2020.00926 (2020).

18. Selvaraju, R. R. *et al.* Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. *Int. J. Comput. Vis.* **128**, 336–359, DOI: 10.1007/s11263-019-01228-7 (2020).

19. Springenberg, J. T., Dosovitskiy, A., Brox, T. & Riedmiller, M. Striving for Simplicity: The All Convolutional Net. *arXiv:1412.6806 [cs]* (2015).

20. Zintgraf, L. M., Cohen, T. S. & Welling, M. A New Method to Visualize Deep Neural Networks. *arXiv:1603.02518 [cs]* (2017).

21. Lee, H., Grosse, R., Ranganath, R. & Ng, A. Y. Unsupervised learning of hierarchical representations with convolutional deep belief networks. *Commun. ACM* **54**, 95–103, DOI: 10.1145/2001269.2001295 (2011).

22. Kingma, D. P. & Welling, M. Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]* (2014).

23. Kingma, D. P. & Welling, M. An Introduction to Variational Autoencoders. *arXiv:1906.02691 [cs, stat]* DOI: 10.1561/2200000056 (2019).

24. Doersch, C. Tutorial on Variational Autoencoders. *arXiv:1606.05908 [cs, stat]* (2021).

25. Goodfellow, I. J. *et al.* Generative Adversarial Networks. *arXiv:1406.2661 [cs, stat]* (2014).

26. Salehi, P., Chalechale, A. & Taghizadeh, M. Generative Adversarial Networks (GANs): An Overview of Theoretical Model, Evaluation Metrics, and Recent Developments. *arXiv:2005.13178 [cs, eess]* (2020).

27. Gui, J., Sun, Z., Wen, Y., Tao, D. & Ye, J. A Review on Generative Adversarial Networks: Algorithms, Theory, and Applications. *arXiv:2001.06937 [cs, stat]* (2020).

28. Papamakarios, G. Neural Density Estimation and Likelihood-free Inference. *arXiv:1910.13233 [cs, stat]* (2019).

29. Kobyzev, I., Prince, S. J. D. & Brubaker, M. A. Normalizing Flows: An Introduction and Review of Current Methods. *arXiv:1908.09257 [cs, stat]* DOI: 10.1109/TPAMI.2020.2992934 (2020).

30. Kingma, D. P. & Dhariwal, P. Glow: Generative Flow with Invertible 1x1 Convolutions. *arXiv:1807.03039 [cs, stat]* (2018).

31. Larsen, A. B. L., Sønderby, S. K., Larochelle, H. & Winther, O. Autoencoding beyond pixels using a learned similarity metric. *arXiv:1512.09300 [cs, stat]* (2016).

32. Nguyen, A., Dosovitskiy, A., Yosinski, J., Brox, T. & Clune, J. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. *arXiv:1605.09304 [cs]* (2016).

33. Dosovitskiy, A. & Brox, T. Generating images with perceptual similarity metrics based on deep networks. *arXiv:1602.02644 [cs]* (2016).

34. Atapattu, C. & Rekabdar, B. Improving the realism of synthetic images through a combination of adversarial and perceptual losses. In *2019 International Joint Conference on Neural Networks (IJCNN)*, 1–7, DOI: 10.1109/IJCNN.2019.8852449 (2019). ISSN: 2161-4407.

35. Schirrmeister, R. T., Chrabaszcz, P., Hutter, F. & Ball, T. Training Generative Reversible Networks. *arXiv:1806.01610 [cs, stat]* (2018).

36. Grathwohl, W., Chen, R. T. Q., Bettencourt, J., Sutskever, I. & Duvenaud, D. FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models. *arXiv:1810.01367 [cs, stat]* (2018).

37. Behrmann, J., Grathwohl, W., Chen, R. T. Q., Duvenaud, D. & Jacobsen, J.-H. Invertible Residual Networks. *arXiv:1811.00995 [cs, stat]* (2019).

38. Baird, L., Smalenberger, D. & Ingkiriwang, S. One-step neural network inversion with PDF learning and emulation. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2, 966–971 vol. 2, DOI: 10.1109/IJCNN.2005.1555983 (2005). ISSN: 2161-4407.

39. Donahue, J., Krähenbühl, P. & Darrell, T. Adversarial Feature Learning. *arXiv:1605.09782 [cs, stat]* (2017).

40. D.O. Hebb: The Organization of Behavior, Wiley: New York; 1949. *Brain Res. Bull.* **50**, 437, DOI: 10.1016/S0361-9230(99)00182-3 (1999).

41. Krotov, D. & Hopfield, J. J. Unsupervised learning by competing hidden units. *Proc. Natl. Acad. Sci.* **116**, 7723–7731, DOI: 10.1073/pnas.1820458116 (2019).

42. Lindsey, J. & Litwin-Kumar, A. Learning to Learn with Feedback and Local Plasticity. *arXiv:2006.09549 [cs, q-bio]* (2020).

43. Isomura, T. & Toyoizumi, T. A Local Learning Rule for Independent Component Analysis. *Sci. Reports* **6**, 28073, DOI: 10.1038/srep28073 (2016).

44. Isomura, T. & Toyoizumi, T. Error-Gated Hebbian Rule: A Local Learning Rule for Principal and Independent Component Analysis. *Sci. Reports* **8**, 1835, DOI: 10.1038/s41598-018-20082-0 (2018).

45. Whittington, J. C. & Bogacz, R. Theories of Error Back-Propagation in the Brain. *Trends Cogn. Sci.* **23**, 235–250, DOI: 10.1016/j.tics.2018.12.005 (2019).

46. Song, Y., Lukasiewicz, T., Xu, Z. & Bogacz, R. Can the Brain Do Backpropagation? Exact Implementation of Backpropagation in Predictive Coding Networks. *Adv. Neural Inf. Process. Syst.* **33**, 22566–22579 (2020).

47. Millidge, B., Tschantz, A., Seth, A. K. & Buckley, C. L. Activation Relaxation: A Local Dynamical Approximation to Backpropagation in the Brain. *arXiv:2009.05359 [cs, q-bio]* (2020).

48. Salvatori, T., Song, Y., Lukasiewicz, T., Bogacz, R. & Xu, Z. Predictive Coding Can Do Exact Backpropagation on Convolutional and Recurrent Neural Networks. *arXiv:2103.03725 [cs]* (2021).

49. Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J. & Hinton, G. Backpropagation and the brain. *Nat. Rev. Neurosci.* **21**, 335–346, DOI: 10.1038/s41583-020-0277-3 (2020).

50. Dauvergne, B. & Hascoët, L. The data-flow equations of checkpointing in reverse automatic differentiation. In *International Conference on Computational Science* (2006).

51. Chen, T., Xu, B., Zhang, C. & Guestrin, C. Training deep nets with sublinear memory cost (2016). 1604.06174.

52. Kumar, R., Purohit, M., Svitkina, Z., Vee, E. & Wang, J. Efficient rematerialization for deep networks. In Wallach, H. *et al.* (eds.) *Advances in Neural Information Processing Systems*, vol. 32 (Curran Associates, Inc., 2019).

53. Sohoni, N. S., Aberger, C. R., Leszczynski, M., Zhang, J. & Ré, C. Low-memory neural network training: A technical report (2019). 1904.10631.

54. Higgins, I. *et al.* beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. (2016).

55. Burgess, C. P. *et al.* Understanding disentangling in β-VAE. *arXiv:1804.03599 [cs, stat]* (2018).

56. Sikka, H., Zhong, W., Yin, J. & Pehlevan, C. A Closer Look at Disentangling in β-VAE. *arXiv:1912.05127 [cs, stat]* (2019).

57. Rezende, D. J. & Viola, F. Taming VAEs. *arXiv:1810.00597 [cs, stat]* (2018).

58. Mao, X. *et al.* Least Squares Generative Adversarial Networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, 2813–2821, DOI: 10.1109/ICCV.2017.304 (2017). ISSN: 2380-7504.

59. Arjovsky, M., Chintala, S. & Bottou, L. Wasserstein GAN. *arXiv:1701.07875 [cs, stat]* (2017).

60. Takida, Y., Liao, W.-H., Uesaka, T., Takahashi, S. & Mitsufuji, Y. Preventing Posterior Collapse Induced by Oversmoothing in Gaussian VAE. *arXiv:2102.08663 [cs]* (2021).

61. Havrylov, S. & Titov, I. Preventing Posterior Collapse with Levenshtein Variational Autoencoder. *arXiv:2004.14758 [cs, stat]* (2020).

62. Lucas, J., Tucker, G., Grosse, R. & Norouzi, M. Don't Blame the ELBO! A Linear VAE Perspective on Posterior Collapse. *arXiv:1911.02469 [cs, stat]* (2019).

63. Johnson, J., Alahi, A. & Fei-Fei, L. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. *arXiv:1603.08155 [cs]* (2016).

64. Ardizzone, L. *et al.* Analyzing Inverse Problems with Invertible Neural Networks. *arXiv:1808.04730 [cs, stat]* (2019).

65. Farias, T. d. S. & Maziero, J. Gradient target propagation. *arXiv:1810.09284 [cs]* (2018).

66. Bengio, Y. How Auto-Encoders Could Provide Credit Assignment in Deep Networks via Target Propagation. *arXiv:1407.7906 [cs]* (2014).

67. Ororbia, A. G., Mali, A., Kifer, D. & Giles, C. L. Conducting Credit Assignment by Aligning Local Representations. *arXiv:1803.01834 [cs, stat]* (2018).

68. Krizhevsky, A., Sutskever, I. & Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. *Adv. Neural Inf. Process. Syst.* **25** (2012).

69. Kingma, D. P. & Ba, J. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]* (2017).

70. Saxe, A. M., McClelland, J. L. & Ganguli, S. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv:1312.6120 [cond-mat, q-bio, stat]* (2014).

71. Hu, W., Xiao, L. & Pennington, J. Provable Benefit of Orthogonal Initialization in Optimizing Deep Linear Networks. *arXiv:2001.05992 [cs, math, stat]* (2020).

72. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B. & Hochreiter, S. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. *arXiv:1706.08500 [cs, stat]* (2018).

73. Seitzer, M. pytorch-fid: FID Score for PyTorch. https://github.com/mseitzer/pytorch-fid (2020). Version 0.1.1.

74. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. & Wojna, Z. Rethinking the Inception Architecture for Computer Vision. 2818–2826 (2016).

75. Lecun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **86**, 2278–2324, DOI: 10.1109/5.726791 (1998).

76. Liu, Z., Luo, P., Wang, X. & Tang, X. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)* (2015).

77. Krizhevsky, A. & Hinton, G. Learning multiple layers of features from tiny images. *Master's thesis, Dep. Comput. Sci. Univ. Tor.* (2009).

Author contributions

All authors contributed equally. All authors reviewed the manuscript.

Competing interests

The authors declare no competing interests.

A Mathematical analysis

This additional material studies the numerical analysis of the technique.

A.1 Convergence of the features

For two fully connected layers, we have $a_j^{(x)} = \hat{a}_j = \sum_i w_{ij} x_i$ and $a_j^{(r)} = \sum_i w_{ij} r_i$. The feature is obtained by optimizing the squared L_2 loss between the two activations:

$$\mathscr{L} = \frac{1}{2}(\hat{a}_j - a_j^{(r)})^2. \tag{9}$$

It follows that r will evolve with the gradient flux:

$$\frac{\partial r_i}{\partial t} = -\frac{\partial \mathscr{L}}{\partial r_i} \;\therefore r_i^t = r_i^{t-1} - \frac{\partial C}{\partial r^{t-1}}. \tag{10}$$

So, we can evaluate r^t at each time step t:

$$r_i^1 = r_i^0 - w_{ij}(\hat{a}_j - w_{ij}r^0) = r_i^0(1-w_{ij}^2) + w_{ij}\hat{a}_j, \tag{11}$$
$$r_i^2 = r_i^1(1-w_{ij}^2) + w_{ij}\hat{a}_j = r_i^0(1-w_{ij}^2)^2 + [1+(1-w_{ij}^2)]w_{ij}\hat{a}_j, \tag{12}$$
$$r_i^3 = r_i^0(1-w_{ij}^2)^3 + [1+(1-w_{ij}^2)+(1-w_{ij}^2)^2]w_{ij}\hat{a}_j, \tag{13}$$
$$r_i^4 = r_i^0(1-w_{ij}^2)^4 + [1+(1-w_{ij}^2)+(1-w_{ij}^2)^2+(1-w_{ij}^2)^3]w_{ij}\hat{a}_j, \tag{14}$$
$$\vdots \tag{15}$$

From the pattern above, we generalize r^t for any time step as follows:

$$r_i^p = r_i^0(1-w_{ij}^2)^p + \sum_{q=0}^{p-1}(1-w^2)^q w_{ij}\hat{a}_j. \tag{16}$$

Under the restriction of $w_{ij}^2 \leq 2$, as t grows we have as a limit case:

$$\lim_{p\to\infty} r_i^0(1-w_{ij}^2)^p = 0 \;\therefore \lim_{p\to\infty}\sum_{q=0}^{p-1}(1-w_{ij}^2)^q w_{ij}\hat{a}_j = \frac{\hat{a}_j}{w_{ij}}. \tag{17}$$

So the loss $\mathscr{L} \to 0$ as $p \to \infty$.

A.2 Convergence of the weights

The L_2 loss, which updates the parameters, is:

$$\mathscr{L} = \frac{1}{2}(x_i - r_i^t)^2 = \frac{1}{2}\left[x_i - r_i^0(1-w_{ij}^2)^t - \sum_{q=0}^{t-1}(1-w_{ij}^2)^q w_{ij}\hat{a}_j\right]^2. \tag{18}$$

For one-shot, $T=1$, training, we have:

$$\mathscr{L} = \frac{1}{2}\left[x_i - r_i^0(1-w_{ij}^2) - w_{ij}\hat{a}_j\right]^2 = \frac{1}{2}\left[x_i - r_i^0(1-w_{ij}^2) - w_{ij}w_{ij}x_i\right]^2. \tag{19}$$

We can rewrite the equation above in vector notation as follows:

$$\mathscr{L} = \frac{1}{2}\left[\mathbf{x} - \mathbf{r}^0(\mathbf{I} - \mathbf{w}^T\mathbf{w}) - \mathbf{w}^T\mathbf{w}\mathbf{x}\right]^2. \tag{20}$$

For any $\mathbf{r}^0$, the equation above has roots for $w_{ii}^2 = 0$. We can see then that the loss is minimal when the weight matrix product is orthogonal, i.e. $\mathbf{w}^T\mathbf{w} = \mathbf{I}$. This has as a consequence that the transposed weight matrix is also its generalized Moore–Penrose inverse or pseudo-inverse $\mathbf{w}^T = \mathbf{w}^{-1}$.

B List of networks

This section lists the networks used for each dataset for feature alignment. The notation $Conv2d(f,k,s,p)$ means output filters f, kernel size kxk, stride s and padding p, while $Linear(n)$ has n fully connected neurons.

Input 1x28x28
Conv2d(32, 3, 1, 1) + LeakyReLU
Conv2d(32, 3, 2, 1) + LeakyReLU
Conv2d(64, 3, 1, 1) + LeakyReLU
Conv2d(64, 3, 2, 1) + LeakyReLU + Flatten
Linear(4096)
Linear(Z)

Table 2. Encoder for MNIST.

Input 3x32x32
Conv2d(32, 3, 1, 1) + LeakyReLU
Conv2d(32, 3, 2, 1) + LeakyReLU
Conv2d(64, 3, 1, 1) + LeakyReLU
Conv2d(64, 3, 2, 1) + LeakyReLU
Conv2d(128, 3, 1, 1) + LeakyReLU
Conv2d(128, 3, 2, 1) + LeakyReLU + Flatten
Linear(2048)
Linear(Z)

Table 3. Encoder for CIFAR-10.

Input 3x32x32
Conv2d(32, 3, 1, 1) + LeakyReLU
Conv2d(32, 3, 2, 1) + LeakyReLU
Conv2d(64, 3, 1, 1) + LeakyReLU
Conv2d(64, 3, 2, 1) + LeakyReLU
Conv2d(128, 3, 1, 1) + LeakyReLU
Conv2d(128, 3, 2, 1) + LeakyReLU
Conv2d(256, 3, 1, 1) + LeakyReLU
Conv2d(256, 3, 2, 1) + LeakyReLU + Flatten
Linear(4096)
Linear(Z)

Table 4. Encoder for CelebA.