



Vanilla Probabilistic Autoencoder

Sebastian Ciobanu

Faculty of Computer Science
Alexandru Ioan Cuza University, Iași, Romania
aciobanusebi@gmail.com

Abstract

The autoencoder, a well-known neural network model, is usually fitted using a mean squared error loss or a cross-entropy loss. Both losses have a probabilistic interpretation: they are equivalent to maximizing the likelihood of the dataset when one uses a normal distribution or a categorical distribution respectively. We trained autoencoders on image datasets using different distributions and noticed the differences from the initial autoencoder: if a mixture of distributions is used the quality of the reconstructed images may increase and the dataset can be augmented; one can often visualize the reconstructed image along with the variances corresponding to each pixel. The code which implements this method can be found at <https://github.com/aciobanusebi/vanilla-probabilistic-ae>.

1 Introduction

Neural networks are machine learning models that have emerged in recent years due to deep learning.

Two standard (supervised) learning tasks are represented by regression and classification. Let $\{(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})\}$ be a training dataset, where n is the number of training instances, $x^{(1)}, \dots, x^{(n)}$ represent the input instances, and $y^{(1)}, \dots, y^{(n)}$ represent the output instances—the labels. If $y^{(i)} \in \mathbb{R}, \forall i \in \{1, \dots, n\}$, then a regression task is performed. The setting $y^{(i)} \in \{1, \dots, K\}, \forall i \in \{1, \dots, n\}$, where K is the number of classes, corresponds to a classification task. In order to carry out the task of predicting the label for a new instance, a neural network constructs a function $f(\cdot)$ which can be either a multi-layer perceptron (MLP) [14, sec. 13.2], a convolutional neural network (CNN) [14, ch. 14], a recurrent neural network (RNN) [14, sec. 15.2] etc. Besides the function $f(\cdot)$, a loss function must be specified. If $y^{(i)} \in \mathbb{R}, \forall i \in \{1, \dots, n\}$, then the mean squared error (MSE) loss is usually utilized:

$$\text{loss}(y_{\text{real}}, y_{\text{predicted}}) = \text{MSE}(y_{\text{real}}, y_{\text{predicted}}) = (y_{\text{real}} - y_{\text{predicted}})^2,$$

where $y_{\text{real}} \in \mathbb{R}$ is the real label of an instance x , and $y_{\text{predicted}} \in \mathbb{R}$ is the predicted label given by $f(x)$. Minimizing this loss is equivalent to maximizing the conditional likelihood [14, sec. 4.6] of the training data when the following assumptions are encountered:

$$Y^{(i)}|x^{(i)} \sim \mathcal{N}(\mu = f(x^{(i)}), \sigma^2 = \text{constant}), \forall i \in \{1, \dots, n\},$$

where $Y^{(i)}$ is the random variable associated with the training label $y^{(i)}$.

If $y^{(i)} \in \{1, \dots, K\}$ or, equivalently, $y^{(i)} \in \{0, 1\}^K$ with $\sum_{k=1}^K y_k^{(i)} = 1, \forall i \in \{1, \dots, n\}$, then the cross-entropy (CE or CH) loss is often used:

$$\begin{aligned} \text{loss}(y_{\text{real}}, y_{\text{predicted}}) &= \text{CH}(y_{\text{real}}, y_{\text{predicted}}) \\ &= \text{CH} \left(\begin{pmatrix} 1 & \dots & K \\ y_{\text{real}_1} & \dots & y_{\text{real}_K} \end{pmatrix}, \begin{pmatrix} 1 & \dots & K \\ y_{\text{predicted}_1} & \dots & y_{\text{predicted}_K} \end{pmatrix} \right) \\ &= - \sum_{k=1}^K y_{\text{real}_k} \log y_{\text{predicted}_k}, \end{aligned}$$

where $y_{\text{real}} \in \{0, 1\}^K$ with $\sum_{k=1}^K y_{\text{real}_k} = 1$ is the real label of an instance x , and $y_{\text{predicted}} \in (0, 1)^K$ with $\sum_{k=1}^K y_{\text{predicted}_k} = 1$ is the predicted label given by $f(x)$. Note that $y_{\text{predicted}}$ is not constrained to contain integer values—0 or 1—, but real values between 0 and 1. Minimizing this loss is equivalent to maximizing the conditional likelihood of the training data when the following assumptions are encountered:

$$Y^{(i)}|x^{(i)} \sim \text{Categorical}(\pi = f(x^{(i)})), \forall i \in \{1, \dots, n\},$$

where $Y^{(i)}$ is the random variable associated with the training label $y^{(i)}$.

Up to this point we have focused on supervised learning. Unsupervised learning, on the other hand, spans other tasks: clustering, anomaly detection, dimensionality reduction etc. The latter one can be realized using an autoencoder—a neural network model. Let $\{x^{(1)}, \dots, x^{(n)}\}$ be a training dataset, where n is the number of training instances. The output of the autoencoder is the reconstruction $x_{\text{reconstructed}}$ of the input x , and this is computed through a neural network which does not allow to learn directly the identity function—e.g. by using a bottleneck autoencoder [14, sec. 20.3.1]. As in the supervised setting, the architecture can be represented by a function $f(\cdot)$, and the loss is either MSE or CH. By exploiting the previously highlighted equivalence between the loss minimization and the likelihood maximization, one can set $Y^{(i)}|x^{(i)}$ to follow distributions different from the normal distribution with constant variance and the categorical distribution. More specifically, we have:

$$X_{\text{reconstructed}}^{(i)}|x^{(i)} \sim \text{Distribution}(\text{parameters} = f(x^{(i)})), \forall i \in \{1, \dots, n\},$$

where $X_{\text{reconstructed}}^{(i)}$ is the random variable associated with the reconstruction of the input $x^{(i)}$: $x_{\text{reconstructed}}^{(i)}$. Unlike the supervised scenario above, the distribution is multivariate, since the input to the autoencoder is usually multi-dimensional and hence the output, as well.

As a result, new models can arise from this perspective, and some of these models are analyzed in this paper.

The structure of this paper is as follows:

- in Section 2 we present the related work;
- Section 3 contains information on our method, methodology, and results;
- in Section 4 the conclusion and ideas for future work are presented.

2 Related Work

Up to our best knowledge, the approach of analyzing the effects of different distributions in

$$X_{\text{reconstructed}}^{(i)}|x^{(i)} \sim \text{Distribution}(\text{parameters} = f(x^{(i)})), \forall i \in \{1, \dots, n\}$$

is not present in the literature in the context of autoencoders, but related ideas do exist.

In regard to single-label supervised learning, we highlight two approaches. The first one regards linear regression with input-dependent variances in the output distribution—i.e. heteroskedastic regression [14, sec. 2.6.3]. More specifically, the assumption is as follows:

$$Y^{(i)}|x^{(i)} \sim \mathcal{N}(\mu = f(x^{(i)})_1, \sigma^2 = f(x^{(i)})_2), \forall i \in \{1, \dots, n\}.$$

Its advantage retains in the fact that confidence intervals can be returned along with the predicted output—the distribution’s mean. The second model regards the mixture of distributions as outputs—e.g. mixture of linear experts [14, sec. 13.6.2.1] and mixture density networks [2] etc.:

$$Y^{(i)}|x^{(i)} \sim \text{Mixture}(\text{parameters} = f(x^{(i)})), \forall i \in \{1, \dots, n\}.$$

Its usage is advantageous when inputs have multiple acceptable outputs—e.g. in image colorization from grayscale images.

As for probabilistic autoencoders, the main contribution in the literature is represented by the variational autoencoder (VAE) [10]. Both the encoder and the decoder are now probabilistic. One advantage is that sampling new data can be performed with a VAE. Moreover, VAEs can be used in clustering when a mixture of distributions is applied on the encoder’s output [8] or can leverage the 2D structure in images via a matrix normal distribution on the encoder’s output [13]. Conditional VAEs [18] take as input also a label corresponding to the instance. If we use an autoencoder instead and set the encoder’s output distribution to a normalizing flow [14, sec. 19.3.6.3], then a probabilistic autoencoder [3] is created. When the encoder is deterministic, and additional regularization is introduced, the autoencoder becomes a Gaussian autoencoder [5]. Conditional probabilities are encountered in discriminative autoencoders which combine autoencoders with classification [16, 17, 12]. With this list of autoencoders, we wanted to emphasize that we are aware of the existence of those algorithms and that their link to our model is only at a high level, being represented by the probabilistic perspective.

3 Method and Experiments

3.1 Method

We start from a bottleneck autoencoder and replace the loss with the negative log-likelihood of our data using different probabilistic distributions in the following structure:

$$X_{\text{reconstructed}}^{(i)}|x^{(i)} \sim \text{Distribution}(\text{parameters} = f(x^{(i)})), \forall i \in \{1, \dots, n\}.$$

We call the resulting model Vanilla Probabilistic Autoencoder. It spans the following scenarios for Distribution:

- MSE = mean squared error
- \mathcal{N}_μ = multivariate normal distribution with the covariance matrix set to the identity matrix:
 $\mathcal{N}(\mu = f(x^{(i)}), \Sigma = I)$; this is mathematically equivalent to MSE
- $\mathcal{N}_{\mu, \text{diag}(\Sigma)}$ = multivariate normal distribution with a diagonal covariance matrix:
 $\mathcal{N}(\mu = f(x^{(i)})_{\text{block } 1}, \Sigma = \text{diag}(f(x^{(i)})_{\text{block } 2}))$

- $\mathcal{N}_{\mu, \Sigma}$ = multivariate normal distribution:
 $\mathcal{N}(\mu = f(x^{(i)})_{\text{block 1}}, \Sigma = f(x^{(i)})_{\text{block 2}})$
- $t_{\text{df}, \mu}$ = multivariate t-distribution with the scale matrix set to the identity matrix:
 $t(\text{df} = f(x^{(i)})_{\text{block 1}}, \mu = f(x^{(i)})_{\text{block 2}}, \Sigma = I)$
- $t_{\text{df}, \mu, \text{diag}(\Sigma)}$ = multivariate t-distribution with a diagonal scale matrix:
 $t(\text{df} = f(x^{(i)})_{\text{block 1}}, \mu = f(x^{(i)})_{\text{block 2}}, \Sigma = \text{diag}(f(x^{(i)})_{\text{block 3}}))$
- $t_{\text{df}, \mu, \Sigma}$ = multivariate t-distribution:
 $t(\text{df} = f(x^{(i)})_{\text{block 1}}, \mu = f(x^{(i)})_{\text{block 2}}, \Sigma = f(x^{(i)})_{\text{block 3}})$
- MAF = masked autoregressive flow [15] with a 10-unit hidden layer with ReLU [6] activations—i.e. a normalizing flow:
 $\text{MAF}(\text{parameters} = f(x^{(i)}))$
- \mathcal{MN}_{μ} = matrix normal distribution with the scale matrices set to the identity matrix:
 $\mathcal{MN}(\mu = f(x^{(i)}), U = I, V = I)$; this is mathematically equivalent to MSE
- $\mathcal{MN}_{\mu, \text{diag}(U, V)}$ = matrix normal distribution with diagonal scale matrices:
 $\mathcal{MN}(\mu = f(x^{(i)})_{\text{block 1}}, U = \text{diag}(f(x^{(i)})_{\text{block 2}}), V = \text{diag}(f(x^{(i)})_{\text{block 3}}))$
- $\mathcal{MN}_{\mu, U, V}$ = matrix normal distribution:
 $\mathcal{MN}(\mu = f(x^{(i)})_{\text{block 1}}, U = f(x^{(i)})_{\text{block 2}}, V = f(x^{(i)})_{\text{block 3}})$
- $T_{\text{df}, \mu}$ = matrix t-distribution with the scale matrices set to the identity matrix:
 $T(\text{df} = f(x^{(i)})_{\text{block 1}}, \mu = f(x^{(i)})_{\text{block 2}}, U = I, V = I)$
- $T_{\text{df}, \mu, \text{diag}(U, V)}$ = matrix t-distribution with diagonal scale matrices:
 $T(\text{df} = f(x^{(i)})_{\text{block 1}}, \mu = f(x^{(i)})_{\text{block 2}}, U = \text{diag}(f(x^{(i)})_{\text{block 3}}), V = \text{diag}(f(x^{(i)})_{\text{block 4}}))$
- $T_{\text{df}, \mu, U, V}$ = matrix t-distribution:
 $T(\text{df} = f(x^{(i)})_{\text{block 1}}, \mu = f(x^{(i)})_{\text{block 2}}, U = f(x^{(i)})_{\text{block 3}}, V = f(x^{(i)})_{\text{block 4}}),$

where $\text{diag}(\cdot)$ receives a vector as input and returns a diagonal matrix with its diagonal set to the input vector.

Furthermore, we also employed a VAE with the following distributions:

$$Z^{(i)} \sim \mathcal{N}(\mu = 0, \Sigma = I)$$

$$Z^{(i)} | x^{(i)} \sim \mathcal{N}(\mu = e(x^{(i)})_{\text{block 1}}, \Sigma = \text{diag}(e(x^{(i)})_{\text{block 2}}))$$

$$X_{\text{reconstructed}}^{(i)} | z^{(i)} \sim \mathcal{N}(\mu = d(z^{(i)})_{\text{block 1}}, \Sigma = \text{diag}(d(z^{(i)})_{\text{block 2}}))$$

where $X_{\text{reconstructed}}^{(i)}$ is the random variable associated with the reconstruction $x_{\text{reconstructed}}^{(i)}$ of the input $x^{(i)}$, $Z^{(i)}$ is the random variable associated with the encoder's output, $z^{(i)}$ is usually a sample from $Z^{(i)} | x^{(i)}$, $e(\cdot)$ is the encoder, and $d(\cdot)$ is the decoder.

Distribution	K	FID↓	L2↓	param.	s/epoch	FID↓	L2↓	param.	s/epoch
-	-	485.073	-	-	-	586.867	-	-	-
MSE	1	520.167	1.585	50992	2.563	689.622	1.451	50992	3.228
\mathcal{N}_μ	1	521.217	1.587	51025	4.325	694.058	1.525	51025	4.331
$\mathcal{N}_{\mu,\text{diag}(\Sigma)}$	1	507.615	5.299	76897	4.667	731.704	4.481	76897	4.715
$\mathcal{N}_{\mu,\Sigma}$	1	447.789	9.808	10205785	60.116	726.986	16.419	10205785	61.522
$t_{\text{df},\mu}$	1	523.926	1.623	51058	5.83	695.567	1.61	51058	7.713
$t_{\text{df},\mu,\text{diag}(\Sigma)}$	1	496.185	5.037	76930	7.655	718.657	3.26	76930	7.738
$t_{\text{df},\mu,\Sigma}$	1	544.819	4.303	10205818	57.132	711.698	3.541	10205818	57.981
MAF	1	524.118	1.869	76090	93.599	699.586	1.609	76090	93.015
\mathcal{MN}_μ	1	521.807	1.668	51025	2.601	696.012	1.526	51025	2.602
$\mathcal{MN}_{\mu,\text{diag}(U,V)}$	1	619.038	5.685	52873	3.503	782.027	4.169	52873	3.369
$\mathcal{MN}_{\mu,U,V}$	1	553.959	6.207	77821	4.824	675.001	14.303	77821	4.902
$T_{\text{df},\mu}$	1	524.907	1.706	51058	5.749	695.669	1.609	51058	7.227
$T_{\text{df},\mu,\text{diag}(U,V)}$	1	558.429	4.571	52906	6.459	739.747	3.519	52906	6.483
$T_{\text{df},\mu,U,V}$	1	585.251	5.61	77854	7.938	746.244	6.307	77854	8.071
VAE	1	444.135	5.412	101984	4.394	652.201	5.608	101984	4.383
MSE	10	-	-	-	-	-	-	-	-
\mathcal{N}_μ	10	500.474	1.163	284170	4.968	667.922	1.046	284170	5.382
$\mathcal{N}_{\mu,\text{diag}(\Sigma)}$	10	465.341	4.575	542890	10.643	686.931	2.708	542890	10.398
$\mathcal{N}_{\mu,\Sigma}$	10	-	-	-	-	-	-	-	-
$t_{\text{df},\mu}$	10	528.223	7.671	284500	51.911	600.937	15.082	284500	51.834
$t_{\text{df},\mu,\text{diag}(\Sigma)}$	10	518.133	4.975	543220	56.917	621.738	16.377	543220	57.293
$t_{\text{df},\mu,\Sigma}$	10	-	-	-	-	-	-	-	-
MAF	10	-	-	-	-	-	-	-	-
\mathcal{MN}_μ	10	501.229	1.208	284170	9.067	667.476	1.043	284170	9.067
$\mathcal{MN}_{\mu,\text{diag}(U,V)}$	10	547.99	4.795	302650	17.817	688.749	2.331	302650	17.909
$\mathcal{MN}_{\mu,U,V}$	10	596.754	5.98	552130	33.195	709.975	10.797	552130	34.137
$T_{\text{df},\mu}$	10	496.058	7.953	284500	53.088	603.4	16.101	284500	44.607
$T_{\text{df},\mu,\text{diag}(U,V)}$	10	544.661	5.149	302980	62.859	612.295	14.122	302980	62.599
$T_{\text{df},\mu,U,V}$	10	578.372	5.677	552460	79.59	683.051	9.389	552460	78.134
VAE	10	-	-	-	-	-	-	-	-

⏟
MNIST
⏟
F-MNIST

Table 1: MLP architecture: FID [7] score on 1000 reconstructed test images, mean squared L2 error (in percentages) between the 1000 test images and the reconstructed ones, the number of parameters of the model, and time measured in seconds per epoch; K is the number of distributions in the mixture. The second row refers to the FID of the actual test images.

3.2 Experiments

We implemented the model in TensorFlow probability [4], a library well-suited for our purpose of working with neural networks and probabilities. The algorithm is a gradient descent version using Adam [9] with the parameters set to the default values in TensorFlow [1].

We used two neural network architectures for the autoencoder:

- MLP with one 32-unit hidden layer, ReLU activation from the input layer to the hidden one, uniform Glorot initialization [14, sec. 13.4.5.1]
- CNN with a convolutional layer with 320 filters of dimension 3×3 , *same* padding, ReLU activation from the input layer to the hidden one, uniform Glorot initialization, then max pooling of size 2×2 , followed by a similar convolutional layer with a suitable number of filters, and then an upsampling layer of dimension 2×2 ; applied only for \mathcal{MN}_μ ,

Distribution	K	FID↓	L2↓	param.	s/epoch	FID↓	L2↓	param.	s/epoch
-	-	485.073	-	-	-	586.867	-	-	-
\mathcal{MN}_μ	1	519.134	8.047	8962	11.24	523.88	8.477	8962	11.287
$\mathcal{MN}_{\mu,\text{diag}(U,V)}$	1	560.765	8.778	8962	11.881	567.215	11.444	8962	11.968
$\mathcal{MN}_{\mu,U,V}$	1	480.298	9.825	11843	13.468	531.369	18.608	11843	13.481
$T_{\text{df},\mu}$	1	528.973	8.074	8962	12.419	537.943	8.521	8962	12.435
$T_{\text{df},\mu,\text{diag}(U,V)}$	1	521.917	8.192	8962	13.013	527.625	8.67	8962	13.052
$T_{\text{df},\mu,U,V}$	1	483.51	47.054	11843	14.601	477.678	15.92	11843	14.555
\mathcal{MN}_μ	10	623.764	8.498	34891	16.566	561.376	10.983	34891	16.621
$\mathcal{MN}_{\mu,\text{diag}(U,V)}$	10	531.622	9.503	34891	27.402	578.51	14.677	34891	27.167
$\mathcal{MN}_{\mu,U,V}$	10	566.882	9.907	63701	43.742	517.706	18.78	63701	42.675
$T_{\text{df},\mu}$	10	642.189	8.682	34891	26.821	585.616	12.052	34891	26.72
$T_{\text{df},\mu,\text{diag}(U,V)}$	10	373.631	11.093	34891	35.38	568.583	17.183	34891	35.742
$T_{\text{df},\mu,U,V}$	10	563.388	17.739	63701	51.653	551.389	14.428	63701	52.031

⏟
MNIST
⏟
F-MNIST

Table 2: CNN architecture: FID score on 1000 reconstructed test images, mean squared L2 error (in percentages) between the 1000 test images and the reconstructed ones, the number of parameters of the model, and time measured in seconds per epoch; K is the number of distributions in the mixture. The second row refers to the FID of the actual test images.

$\mathcal{MN}_{\mu,\text{diag}(U,V)}$, $\mathcal{MN}_{\mu,U,V}$, $T_{\text{df},\mu}$, $T_{\text{df},\mu,\text{diag}(U,V)}$, $T_{\text{df},\mu,U,V}$; in this context, there are no Dense¹ layers.

We implement the parameter constraints as follows:

- strictly positive p : $p = 10^{-6} + \text{softplus}(p)$ [14, sec. 2.6.3];
- positive definite matrix A : $A = \text{FillScaleTriL}(A)$ ².

Given a distribution, we obtain an image by returning the mean—for the normal distributions—, a mode or an approximation of it—for the t distributions—, or an approximation of the mean—for the MAF model by averaging 50 samples. The returned image is post-processed by clipping its values such that all the pixels are in the interval $[0, 1]$.

The number of epochs is 20, the batch size is set to 128, and the shuffle buffer size is 1024. If the result is not a number (nan), then other runs are executed.

We applied this approach on the following image datasets:

- MNIST [11]: 60000 training + 10000 testing grayscale images, 28×28 or 784 pixels, 10 classes, pixel value $\in [0, 1]$
- Fashion-MNIST (F-MNIST) [20]: 60000 training + 10000 testing grayscale images, 28×28 or 784 pixels, 10 classes, pixel value $\in [0, 1]$.

Since these datasets are grouped into 10 classes, we also set the output of the autoencoder to be a mixture of 10 distributions, as we will see in the results.

Since the results contain time data, we include below the specifications of the machine that carried out the experiments: Windows 10 OS, Lenovo Legion Y720 Laptop, NVIDIA GeForce GTX 1060 6 GB GPU, Intel(R) Core(TM) i7-7700HQ CPU @ 2.80Hz, 32.0 GB RAM.

¹https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense

²https://www.tensorflow.org/probability/api_docs/python/tfp/bijectors/FillScaleTriL




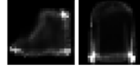





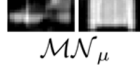
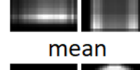
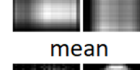
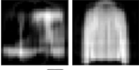


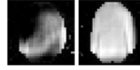
MLP architecture					
 Real image	 MSE	 mean \mathcal{N}_μ	 variance mean $\mathcal{N}_{\mu,\text{diag}(\Sigma)}$	 variance mean $\mathcal{N}_{\mu,\Sigma}$	 mode $t_{\text{df},\mu}$
 variance: nan mode $t_{\text{df},\mu,\text{diag}(\Sigma)}$	 variance: nan mode $t_{\text{df},\mu,\Sigma}$	 \sim mean MAF	 mean \mathcal{MN}_μ	 variance mean $\mathcal{MN}_{\mu,\text{diag}(U,V)}$	 variance mean $\mathcal{MN}_{\mu,U,V}$
 mode $T_{\text{df},\mu}$	 \sim variance mode $T_{\text{df},\mu,\text{diag}(U,V)}$	 \sim variance mode $T_{\text{df},\mu,U,V}$	 mean VAE		

Figure 1: Reconstructing images using different models (1). Where possible, we included the variances which were also clipped to be in $[0, 1]$, although dividing all the variances by the maximum variance is an alternative; we computed the variance in the case of matrix distributions by multiplying the row and column variances: this is mathematically correct for the normal matrix distribution, but just an approximation for the matrix t-distribution; low variances are darker and high variances are lighter; if the variance is nan, then the covariance matrix of the corresponding distribution does not exist.

3.3 Results

We present the results in Tables 1–2 and Figures 1–4. The figures reflect only the F-MNIST dataset for brevity. Our analysis is twofold.

Firstly, we take into consideration the mutual characteristics of our models, of the MSE autoencoder, and of the VAE. Tables 1 and 2 contain information on the number of parameters, fitting time expressed in seconds per epoch, the Fréchet inception distance (FID) scores of the reconstruction of 1000 new/test images which hopefully express the visual quality of the reconstructions, and the mean squared L2 reconstruction error for the 1000 test images. The best FID scores are obtained for the VAE, $\mathcal{N}_{\mu,\text{diag}(\Sigma)}$, $t_{\text{df},\mu}$, $\mathcal{MN}_{\mu,U,V}$, $T_{\text{df},\mu,U,V}$, and $T_{\text{df},\mu,\text{diag}(U,V)}$ models. Some of those scores even surpass the FID scores of the real data, but the reconstructions in Figures 1–2 suggest that the reconstructions may not truly reproduce the input and they still look natural—e.g. see $t_{\text{df},\mu}$ in the first row in Figure 2—or not—e.g. see $\mathcal{MN}_{\mu,U,V}$ in the last row in Figure 2. Figures 1–2 also reflect that the mixture with 10 distributions can obtain the best visual results among the models. As a result, for the chosen datasets the FID scores do not necessarily express image quality. The best L2 errors are obtained for the


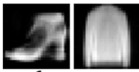



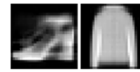
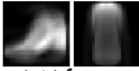
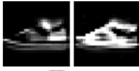

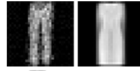
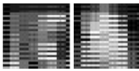
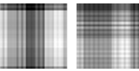
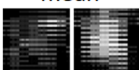
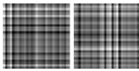

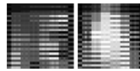
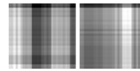
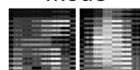
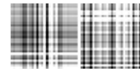

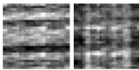
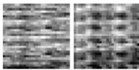
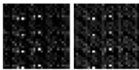
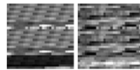
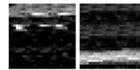
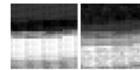
mixture with 10 distributions; MLP architecture					
mean  \mathcal{N}_μ	mean  $\mathcal{N}_{\mu,\text{diag}(\Sigma)}$	\sim mode  $t_{\text{df},\mu}$	\sim mode  $t_{\text{df},\mu,\text{diag}(\Sigma)}$	mean  \mathcal{MN}_μ	mean  $\mathcal{MN}_{\mu,\text{diag}(U,V)}$
mean  $\mathcal{MN}_{\mu,U,V}$	\sim mode  $T_{\text{df},\mu}$	\sim mode  $T_{\text{df},\mu,\text{diag}(U,V)}$	\sim mode  $T_{\text{df},\mu,U,V}$		
CNN architecture					
mean  \mathcal{MN}_μ	variance  mean  $\mathcal{MN}_{\mu,\text{diag}(U,V)}$	variance  mean  $\mathcal{MN}_{\mu,U,V}$	mode  $T_{\text{df},\mu}$	\sim variance  mode  $T_{\text{df},\mu,\text{diag}(U,V)}$	\sim variance  mode  $T_{\text{df},\mu,U,V}$
mixture with 10 distributions; CNN architecture					
mean  \mathcal{MN}_μ	mean  $\mathcal{MN}_{\mu,\text{diag}(U,V)}$	mean  $\mathcal{MN}_{\mu,U,V}$	\sim mode  $T_{\text{df},\mu}$	\sim mode  $T_{\text{df},\mu,\text{diag}(U,V)}$	\sim mode  $T_{\text{df},\mu,U,V}$

Figure 2: Reconstructing images using different models (2). Where possible, we included the variances which were also clipped to be in $[0, 1]$, although dividing all the variances by the maximum variance is an alternative; we computed the variance in the case of matrix distributions by multiplying the row and column variances: this is mathematically correct for the normal matrix distribution, but just an approximation for the matrix t-distribution; low variances are darker and high variances are lighter.

MSE autoencoders—or their equivalents: \mathcal{N}_μ and \mathcal{MN}_μ —in the non-mixture models; this is as expected because those models use the L2 loss at the training stage. Even in the mixture setup, the equivalents of the MSE autoencoders usually obtain the lowest L2 scores.

The CNN architecture cannot learn significant reconstructions—we also tried with 32 hidden filters and the results were approximately the same—, probably because the output of the neural network contains extra numbers that are not used in the loss—we did not use Dense layers, so we adjusted the number of output filters to obtain a number of outputs greater than or equal to the required number of values to be fed into the probabilistic distribution. The $\mathcal{N}_{\mu,\Sigma}$ and $t_{\text{df},\mu,\Sigma}$ models have the greatest number of parameters, and this could create problems if there is not enough memory available. Moreover, the MSE autoencoder is the fastest, and the MAF model is the slowest. The running time for the mixtures is usually less than 10 times the training time of fitting the corresponding distribution alone. Figure 3 includes two ways of sampling new data points. The first row seems better than the second one. The third row contains t-SNE representations which do not manage to perfectly separate the dataset into the initial classes in the latent space, but it tries to; the t-SNE plots are relatively similar.

Secondly, we highlight the unique features of our models. Specifically, we focus on displaying also the variance in an image and creating new data instances from the mixture models. Figures

	mixture with 10 distributions					
	$t_{df,\mu}$	MAF	\mathcal{N}_μ	$\mathcal{N}_{\mu,\text{diag}(\Sigma)}$	\mathcal{MN}_μ	$\mathcal{MN}_{\mu,\text{diag}(U,V)}$
1						
2						
3						

Figure 3: Six MLP models, three rows: row 1 contains samples by sampling from the standard normal distribution in the space of the encoder’s output—or latent space—and propagating them through the decoder; row 2 contains sampled images from the returned distribution when given an input image; row 3 contains t-SNE [19] 2D representations of the latent features of the testing data, and the color represents the label of the data point.

1–2 also contain variances associated with the output image; these are computed from the output distribution. These could be informative when deciding if the pixels in the reconstructed image are those that were present in the initial image. Given an input image, if the output distribution is a mixture, one can compute the means/modes of each distribution in the mixture. This idea is presented in Figure 4. Most of these images have high visual quality and therefore can be used to augment the dataset if needed. Obviously, one can select only those images whose log-probability/log-density passes a preset threshold. We should however remember that the mixture models take a longer time to train than the non-mixture counterparts.

4 Conclusion and Future Work

We proposed to replace the usual losses—i.e. MSE and CH—in the autoencoder with negative log-likelihood losses. Those likelihood losses depend on a specific distribution. The range of the distributions we chose spans from multivariate normal, multivariate t, matrix normal, and matrix t, each with different settings, to mixtures and a normalizing flow. The results suggest that our mixture models can be taken into account as augmentation techniques. The models with full scale/covariance matrices have more parameters and require more memory/epochs than those with diagonal scale/covariance matrices, and this may represent a good reason to choose the diagonal case. In the mixture setting, even with constant scale/covariance matrices, good visual results can be obtained. Pixelwise variances can also be visualized when scale/covariance matrices exist.

As for future work, we can convert a mixture of normal distributions to a value via a mode—not the mean—, we can assess the quality of images via other metrics than FID or L2, we can apply our method to RGB images and non-image datasets. The list of distributions may also include the factor analysis model [14, sec. 20.2]. One can extend our idea to other



Figure 4: Four mixture models (MLP): given the input image at the top, the mean of each distribution in the mixture and its mixture log-density are represented.

image-to-image translation tasks—e.g. image segmentation. Bottleneck autoencoders can be replaced with denoising autoencoders. Moreover, other hyperparameter values can be further explored.

References

- [1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. TensorFlow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.
- [2] Christopher M Bishop. Mixture density networks. 1994.
- [3] Vanessa Böhm and Uroš Seljak. Probabilistic auto-encoder. *arXiv preprint arXiv:2006.05479*, 2020.
- [4] Joshua V Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, and Rif A Saurous. TensorFlow distributions. *arXiv preprint arXiv:1711.10604*, 2017.
- [5] Jarek Duda. Gaussian autoencoder. *arXiv preprint arXiv:1811.04751*, 2018.
- [6] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

- [7] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. *Advances in Neural Information Processing Systems*, 30, 2017.
- [8] Zhuxi Jiang, Yin Zheng, Huachun Tan, Bangsheng Tang, and Hanning Zhou. Variational deep embedding: An unsupervised and generative approach to clustering. *arXiv preprint arXiv:1611.05148*, 2016.
- [9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [10] Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- [11] Yann LeCun, Corinna Cortes, and CJ Burges. MNIST handwritten digit database, 2010.
- [12] Hung-Shin Lee, Yu-Ding Lu, Chin-Cheng Hsu, Yu Tsao, Hsin-Min Wang, and Shyh-Kang Jeng. Discriminative autoencoders for speaker verification. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5375–5379. IEEE, 2017.
- [13] Jinghua Li, Huixia Yan, Junbin Gao, Dehui Kong, Lichun Wang, Shaofan Wang, and Baocai Yin. Matrix-variate variational auto-encoder with applications to image process. *Journal of Visual Communication and Image Representation*, 67:102750, 2020.
- [14] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2021.
- [15] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. *arXiv preprint arXiv:1705.07057*, 2017.
- [16] Angshuman Paul, Angshul Majumdar, and Dipti Prasad Mukherjee. Discriminative autoencoder. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 3049–3053. IEEE, 2018.
- [17] Sebastien Razakarivony and Frédéric Jurie. Discriminative autoencoders for small targets detection. In *2014 22nd International Conference on Pattern Recognition*, pages 3528–3533. IEEE, 2014.
- [18] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. *Advances in Neural Information Processing Systems*, 28:3483–3491, 2015.
- [19] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(11), 2008.
- [20] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.