An Iterative Closest Points Approach to Neural Generative Models

Joose Rajamäki [1]   Perttu Hämäläinen [1]

Abstract

We present a simple way to learn a transformation that maps samples of one distribution to the samples of another distribution. Our algorithm comprises an iteration of 1) drawing samples from some simple distribution and transforming them using a neural network, 2) determining pairwise correspondences between the transformed samples and training data (or a minibatch), and 3) optimizing the weights of the neural network being trained to minimize the distances between the corresponding vectors. This can be considered as a variant of the Iterative Closest Points (ICP) algorithm, common in geometric computer vision, although ICP typically operates on sensor point clouds and linear transforms instead of random sample sets and neural nonlinear transforms. We demonstrate the algorithm on simple synthetic data and MNIST data. We furthermore demonstrate that the algorithm is capable of handling distributions with both continuous and discrete variables.

1. Introduction

Learning transformations between distributions has recently been studied extensively because it allows drawing samples from a simple distribution and mapping them to a more complex distribution of interest. To train neural networks to map distributions to other distributions is usually performed by generative adversarial network training (GAN) (Goodfellow et al., 2014). GANs are, however, notoriously difficult to train because of the unstable dynamics between two competing networks.

We present a simple and stable algorithm to train transformations between distributions such as the one demonstrated in Figure 1.In contrast to GANs, our training method involves just one network. The contribution of this paper is the training algorithm that yields robust training of neural networks for mapping between distributions. Our algorithm comprises an iteration of 1) drawing samples from some simple distribution and transforming them using a neural network, 2) determining pairwise correspondences between the transformed samples and training data (or a minibatch), and 3) optimizing the weights of the neural network being trained to minimize the distances between the corresponding vectors. This can be considered as a variant of the Iterative Closest Points (ICP) algorithm, common in geometric computer vision, although ICP typically operates on sensor point clouds and linear transforms instead of random sample sets and neural nonlinear transforms.

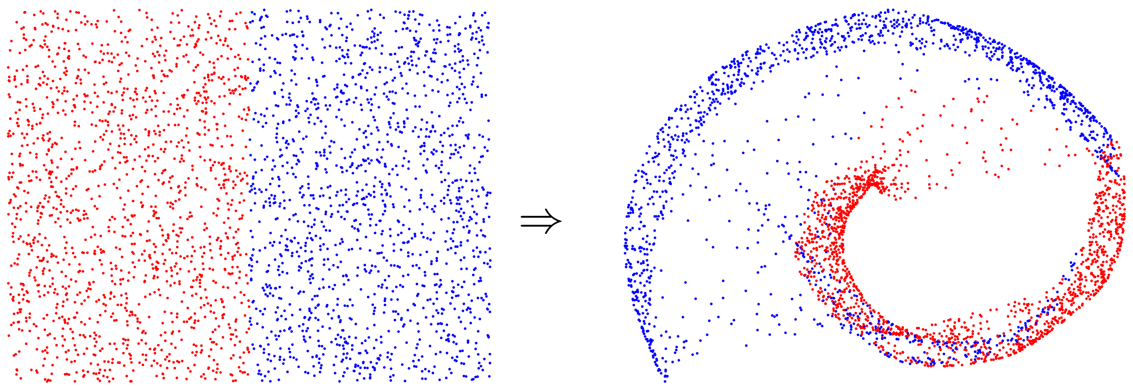
Our algorithm works well also with conditioned data generation and we can also use it to map mixed distributions to discrete ones. To the best of our knowledge, this is the first algorithm to demonstrate mappings from a mixed distribution to a discrete distribution using just one neural network.

An implementation of the algorithm is available in Github: https://github.com/JooseRajamaeki/ICP.

2. Related methods

One of the most succesful approaches in generative models are generative adversarial networks (Goodfellow et al., 2014). Generative adversarial training of neural networks involves training a generator network to map some easy-to-sample distribution to another distribution. The training of GANs also involves training a discriminator network that tries to tell apart the data from the target distribution and the data generated by the generator network. This idea is simple and brilliant but the training involves of technical challenges. The most notable challenge is coordinating the speed at which each of the networks is trained. It happens easily that the discriminator network quickly learns to tell apart the true data and the generated data after which it does not provide a strong gradient to the generator network. These problems have been alleviated to some extent by Wasserstein-GANs (Arjovsky et al., 2017) and their extensions (Gulrajani et al., 2017). We do not have any problems arising from the unstable dynamics of two networks. Furthermore, since we are minimizing a distance measure, we have in all sensible cases a strong gradient.

[1]Aalto University, Helsinki, Finland. Correspondence to: Joose Rajamäki <joose.rajamaki@aalto.fi, joose.rajamaki@gmail.com>, Perttu Hämäläinen <perttu.hamalainen@aalto.fi>.



(a) Samples from the origin distribution.

(b) Generated samples.

Figure 1: A two dimensional uniform distribution is transformed to a swiss roll with added noise.

Optimizing latent variable space can be performed to obtain fast to train algorithm with results that are competitive with GANs (Bojanowski et al., 2017). The algorithm trains only one neural network, like we do. The algorithm learns the latent space, though, i.e. one is not free to choose the sampling distribution but it is instead intractable and given by the training algorithm.

Using bijective mappings has been used to build models for density estimation and sampling (Dinh et al., 2015; 2016). They map a distribution to a latent space with a simple structure. Density estimation in the latent space is straightforward and sampling can be performed effectively as ancestral sampling. Our training algorithm does not impose any restrictions on the model being trained. For instance we can train neural networks to map distributions from continuous to discrete, for which one cannot train bijective mappings.

Variational autoencoders (VAE) (Kingma & Welling, 2014) are another successful approach to train generative models. They utilize the reparametrization trick to train an encoder and a decoder by backpropagation. The encoder transforms the data points to the latent space and the decoder transforms the latent space variables back to the data space. In contrast to our algorithm, the variational autoencoders also involve training two networks. Furthermore, the reparametrization trick sets restrictions to the distributions involved, such as demands for continuity. There are recent advances in extending the variational autoencoders to discrete distributions (Jang et al., 2017; Maddison et al., 2016). This involves further approximations, though, whereas our algorithm extends naturally to discrete distributions as we demonstrate in Section 4.3. Our algorithm offers near complete freedom to choose the distributions involved and the associated model. In Section 4 we use a mixed distribution with discrete and continuous variables as the input ”noise” distribution.

The methods described above involve training a network to map from one distribution to other. Also other generative models and ways to use neural networks exist. For example Bordes et al. (2017) present a denoising approach with factorial start distribution. Bordes et al. (2017) also offer a comprehensive review of the techniques used in generative modeling.

Our method is reminiscent of the Iterative Closest Points (ICP) algorithm (Chen & Medioni, 1991; Besl & McKay, 1992), common in computer vision geometry processing. ICP alternates between 1) computing pairwise correspondences between two sets of vectors, and 2) optimizing a transform that minimizes the distance between corresponding vectors. In ICP, the vector sets are typically point clouds from some sensor(s), whereas in our case, one set is a batch of training data, and the other set is drawn from some simple distribution and transformed by a neural network to the training data space. In our case, the sets are also drawn independently for each iteration whereas they stay the same in each ICP iteration. Furthermore, in ICP the optimized transform is usually a simple geometric one, for which a closed-form solution exist. In contrast to that, we optimize the neural network weights through backpropagation with the sum of pairwise distances as the loss.

3. Methods

This section presents our algorithms. As stated before we have developed an algorithm for learning a mapping to transform samples drawn from one distribution to another distribution. We can also learn a mapping conditioned on a subset of the training data variables.

3.1. Notational conventions and the problem setting

$\mathbf{x}$ are noise from origin or "noise" distribution $\mathcal{D}_{\text{origin}}$, $\mathbf{y}$ are the values from the target distribution $\mathcal{D}_{\text{target}}$, i.e. the training data, and $\mathbf{z}$ denote the conditioning part of the target distribution $\mathcal{D}_{\text{target}}$, if such values are needed.

Our algorithm necessitates a meaningful distance $d(\mathbf{y}_1, \mathbf{y}_2)$ for the space in which the data points $\mathbf{y}_1$ and $\mathbf{y}_2$ reside. We furthermore assume that we have a trainable function approximator f that is continuous. In our tests the function approximator was chosen to be a neural net. We use the circumflex to denote the variables that we get by mapping some variables with f, for example $\widehat{y} = f(x)$.

Our aim is to build an algorithm that works in the same manner as GANs. Given samples $\{\mathbf{y}_0, \mathbf{y}_1, ..., \mathbf{y}_N\}$ we wish to train the mapping f such that it produces "predictions" $\widehat{\mathbf{y}}$ that are indistinguishable from the given samples. We assume that the probability density function or the probability mass function of the distribution being approximated is not known. When it is known, efficient algorithms to train f exist (Liu & Wang, 2016).

3.2. The iterative closest points for distribution training algorithm

Our method is described as pseudocode in Algorithm 1. Its conditioned version is presented in Algorithm 2. The algorithms work by choosing random data point from the true data and finding the closest mapped point. This is performed incrementally such that the mapped points that have already been matched to some data point are not considered in the matching procedure. When all the data points have been matched, the mapping f is trained with supervised learning. Figure 2 illustrates the algorithm.

Because the mapping is continuous, similar points in the input space map to similar points in the output space, i.e. $f(\mathbf{x}_i) \approx f(\mathbf{x}_i + \varepsilon)$. Thus, even if some points get matched poorly, the well-matched points in their surrounding will draw them to the right directions. This implies that f can be thought of as a complicated weighting function of a self-organizing map (Kohonen, 1982).

4. Results

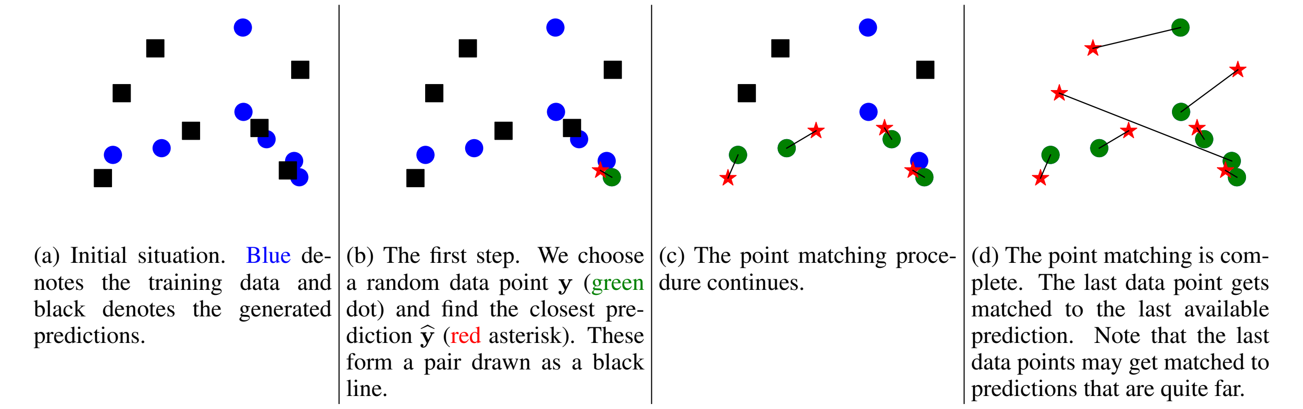
This section presents some results obtained with the algorithm. All of the results were produced with a fully connected neural network with three hidden layers as the mapping f. The units of the hidden layers were chosen to be bipolar (Eidnes & Nøkland, 2017) scaled exponential linear units (Klambauer et al., 2017). The neural neural net in Section 4.2 had 300 units per hidden layer. Otherwise the networks had 50 units per hidden layer. The supervised training was performed by ADAM algorithm (Kingma &

Algorithm 1 The iterative closest points for distribution training algorithm

```
 1: repeat
 2:   Sample set $\mathcal{D}_{\text{batch\_origin}} = \{\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_N\}$ from $\mathcal{D}_{\text{origin}}$
 3:   Sample set $\mathcal{D}_{\text{batch\_target}} = \{\mathbf{y}_0, \mathbf{y}_1, ..., \mathbf{y}_N\}$ from $\mathcal{D}_{\text{target}}$
 4:   Initialize $\mathcal{D}_{\text{pairs}} = \emptyset$
 5:   for all $\mathbf{x}_i \in \mathcal{D}_{\text{batch\_origin}}$ do
 6:     Perform the mapping $\widehat{\mathbf{y}}_i = f(\mathbf{x}_i)$
 7:     Add pair $(\mathbf{x}_i, \widehat{\mathbf{y}}_i)$ to $\mathcal{D}_{\text{pairs}}$
 8:   end for
 9:   Initialize $\mathcal{D}_{\text{ordered}} = \emptyset$
10:   for all $\mathbf{y}_i \in \mathcal{D}_{\text{batch\_target}}$ do
11:     Find index $j = \arg\min d(\mathbf{y}_i, \widehat{\mathbf{y}}_j)$
12:     Add pair $(\mathbf{x}_j, \mathbf{y}_i)$ to $\mathcal{D}_{\text{ordered}}$
13:     Remove pair $(\mathbf{x}_j, \widehat{\mathbf{y}}_j)$ from $\mathcal{D}_{\text{pairs}}$
14:   end for
15:   Train $f$ using $\mathcal{D}_{\text{ordered}}$ by running one epoch of supervised learning
16: until Convergence
```

Algorithm 2 The conditioned iterative closest points for distribution training algorithm

```
 1: repeat
 2:   Sample set $\mathcal{D}_{\text{batch\_origin}} = \{\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_N\}$ from $\mathcal{D}_{\text{origin}}$
 3:   Sample set
      $\mathcal{D}_{\text{batch\_target}} = \left\{ \begin{bmatrix}\mathbf{z}_0\\ \mathbf{y}_0\end{bmatrix}, \begin{bmatrix}\mathbf{z}_1\\ \mathbf{y}_1\end{bmatrix}, ..., \begin{bmatrix}\mathbf{z}_N\\ \mathbf{y}_N\end{bmatrix} \right\}$
      from $\mathcal{D}_{\text{target}}$
 4:   Initialize $\mathcal{D}_{\text{pairs}} = \emptyset$
 5:   for $i = 1...N$ do
 6:     Perform the mapping $\begin{bmatrix}\widehat{\mathbf{z}}_i\\ \widehat{\mathbf{y}}_i\end{bmatrix} = f\left(\begin{bmatrix}\mathbf{z}_i\\ \mathbf{x}_i\end{bmatrix}\right)$
 7:     Add pair $\left(\begin{bmatrix}\mathbf{z}_i\\ \mathbf{x}_i\end{bmatrix}, \begin{bmatrix}\mathbf{z}_i\\ \widehat{\mathbf{y}}_i\end{bmatrix}\right)$ to $\mathcal{D}_{\text{pairs}}$
 8:   end for
 9:   Initialize $\mathcal{D}_{\text{ordered}} = \emptyset$
10:   for all $\mathbf{y}_i \in \mathcal{D}_{\text{batch\_target}}$ do
11:     Find index $j = \arg\min d\left(\begin{bmatrix}\mathbf{z}_i\\ \mathbf{y}_i\end{bmatrix}, \begin{bmatrix}\mathbf{z}_j\\ \widehat{\mathbf{y}}_j\end{bmatrix}\right)$
12:     Add pair $\left(\begin{bmatrix}\mathbf{z}_i\\ \mathbf{x}_j\end{bmatrix}, \begin{bmatrix}\mathbf{z}_i\\ \mathbf{y}_i\end{bmatrix}\right)$ to $\mathcal{D}_{\text{ordered}}$
13:     Remove pair $\left(\begin{bmatrix}\mathbf{z}_j\\ \mathbf{x}_j\end{bmatrix}, \begin{bmatrix}\mathbf{z}_j\\ \widehat{\mathbf{y}}_j\end{bmatrix}\right)$ from $\mathcal{D}_{\text{pairs}}$
14:   end for
15:   Train $f$ using $\mathcal{D}_{\text{ordered}}$ by running one epoch of supervised learning
16: until Convergence
```



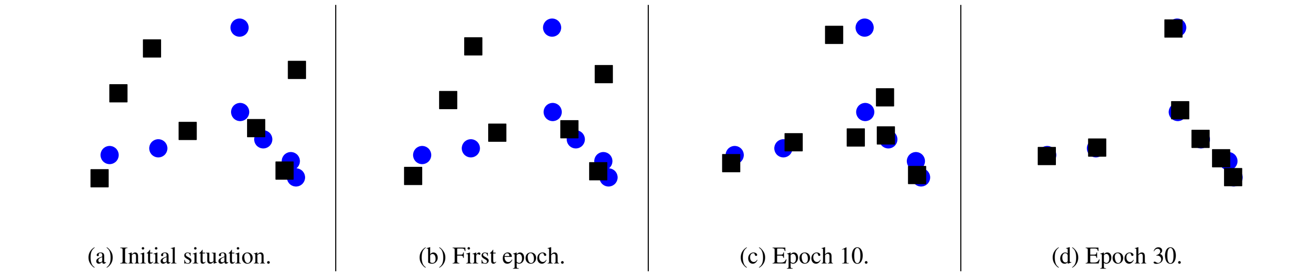
(a) Initial situation. Blue denotes the training data and black denotes the generated predictions.

(b) The first step. We choose a random data point $\mathbf{y}$ (green dot) and find the closest prediction $\widehat{\mathbf{y}}$ (red asterisk). These form a pair drawn as a black line.

(c) The point matching procedure continues.

(d) The point matching is complete. The last data point gets matched to the last available prediction. Note that the last data points may get matched to predictions that are quite far.

Figure 2: An illustrative example of the idea of the algorithm. The predictions $\widehat{\mathbf{y}}$ (black squares) are matched to the actual data (blue dots).



(a) Initial situation.

(b) First epoch.

(c) Epoch 10.

(d) Epoch 30.

Figure 3: An illustrative example of how the predictions $\widehat{\mathbf{y}}$ (black squares) will start to get closer to a point of actual data (blue dots).

Ba, 2015). We furthermore clipped the gradients by clamping each dimension between -0.1 and 0.1.

To highlight the flexibility of our the algorithm, we used a mixed value distribution as the origin distribution $\mathcal{D}_{\text{origin}}$ in all our tests. Apart from Figures 1 and 7, of the N dimensions the first $N/2$ dimensions were sampled from a Bernoulli distribution with $p = 0.5$ and the last $N/2$ dimensions were sampled from the continuous uniform distribution. In Figures 1 and 7 the noise dimension N is obviously 2, for the simple synthetic examples of Section 4.1 N was 6, and otherwise it was 20.

4.1. Low-dimensional examples

Figure 4 presents a three component Gaussian mixture model and Figure 5 shows the same model with the model being conditioned by the x-axis variable. The distance metric used here was the Euclidean distance.

These examples illustrate how the algorithm is able to map a distribution lacking modalities to a three-modal distribution. Some stray data points remain between the distribution components because of the continuous mapping f. Figure shows 6 a sinusoid with added Gaussian noise. Here we observe no problems with the algorithm, as the target distribution does not exhibit multi-modality that would be in contrast to the input distribution. Figure 7 shows the same sinusoid based distribution being approximated with an adversarially chosen two dimensional mixed distribution with a discrete and a continuous variable. This figure also makes evident how the points that are close in the input space map close to each other in the output space.

In these low-dimensional examples we had 1,000 data points and used a batch size of 500 in the training. The supervised training was done with minibatch size 100.

4.2. MNIST data

We also applied the method to the MNIST data (LeCun et al.). Some characters obtained this way are presented in Figure 8. Also surprisingly to us, a simple fully connected neural network can produce all the characters with variation in each character.

The characters in Figure 8 have been generated conditionally, all by the same neural network. In this example we also used the Euclidean distance:

$$d\left(\begin{bmatrix} z_1 \\ \mathbf{y}_1 \end{bmatrix}, \begin{bmatrix} z_2 \\ \mathbf{y}_2 \end{bmatrix}\right) = (z_1 - z_2)^2 + (\mathbf{y}_1 - \mathbf{y}_2)^{\mathrm{T}} (\mathbf{y}_1 - \mathbf{y}_2). \quad (1)$$

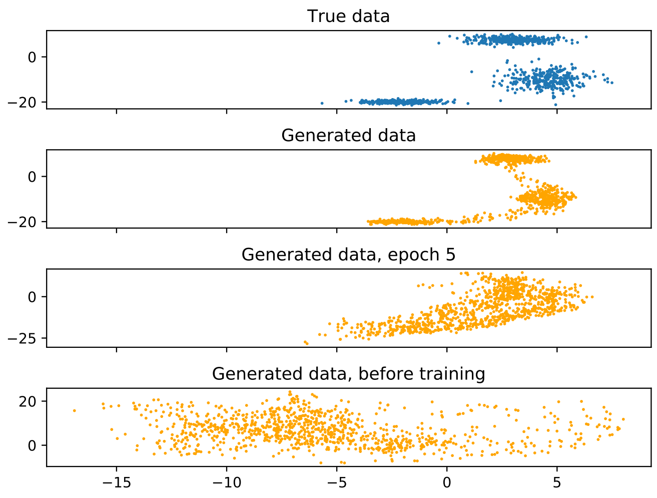


Figure 4: A simple data set of three Gaussian distributions being approximated. The initial distribution has a very small overlap with the data, especially with the uppermost Gaussian. The algorithm improves fast and already after 5 epochs the sampled data starts to assume the form of the true data.

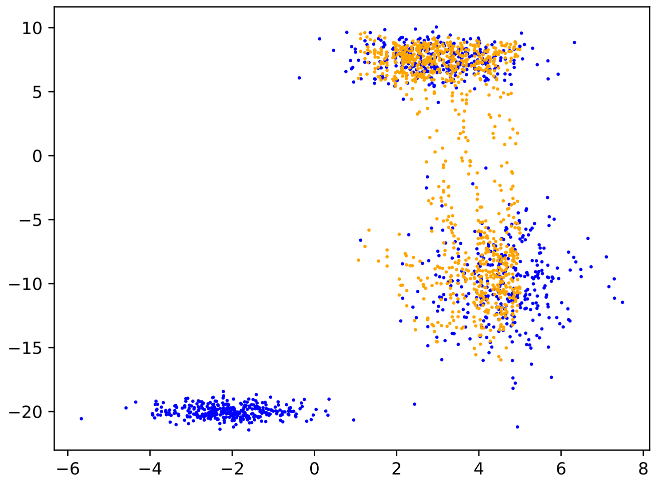


Figure 5: Conditioned data generation being demonstrated with a simple data set of three Gaussians. The x-axis value is selected from the interval [1,5] and the algorithm is asked to predict the missing y-axis value. Blue denotes the ground truth data and orange is generated data. Some stray data points remain between the Gaussians but generally the algorithm captures the shape of the distribution well.

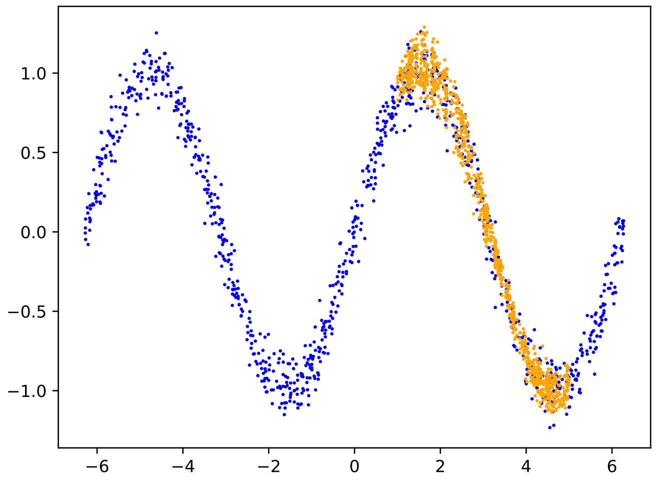
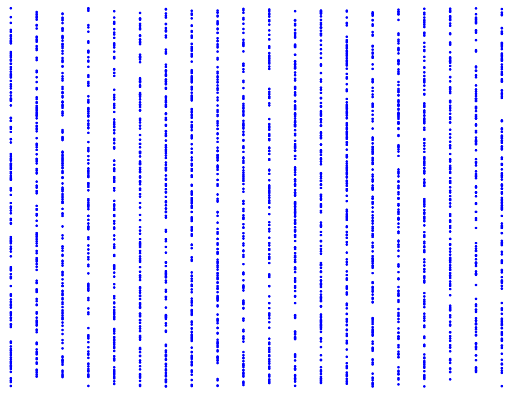


Figure 6: Demonstrating conditioned data generation with a sinusoid that has Gaussian noise added in the y-axis direction. The x-axis value is selected from the interval [1,5] and the algorithm is asked to predict the missing y-axis value. Blue denotes the ground truth data and orange is generated data.

The Euclidean distance is not particularly good for image data. This is why it was surprising to see the algorithm perform quite well with a network architecture (fully connected) and a distance measure that are neither particularly good for the task at hand.

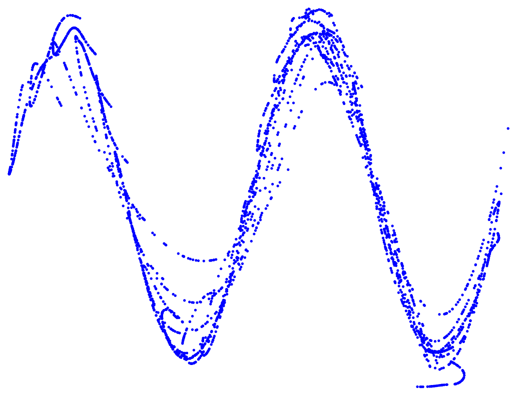
The neural net starts to get some characters right already after the first epoch. Especially the characters that are distinctive from the others, such as zeros find their form early. In the early phase of the training the algorithm seems to produce characters that are mixtures of two different characters. A few examples are shown in Figure 10. As the training progresses, these mixed forms become increasingly rare.

The training set of MNIST data contains 60,000 data points. We used the batch size 10,000 in the matching algorithm and the supervised training was performed with the minibatch size of 100. We also tried using a small minibatch size of 100 both in the matching algorithm and in the supervised learning, i.e. a simplified version of the algorithm where the correspondence finding batch and supervised training minibatch are the same and only one gradient step is taken for a set of corresponding sample pairs. The results are shown in Figure 9. The algorithm works to some extent but the results have smaller variation and the neural network produces some quite faint characters.



(a) Samples from the origin distribution.

$\Downarrow$



(b) Generated samples.

Figure 7: A two dimensional mixed continuous and discrete distribution is transformed to a sinusoid with added Gaussian noise. The stripy pattern of the input distribution translates also to the output distribution.

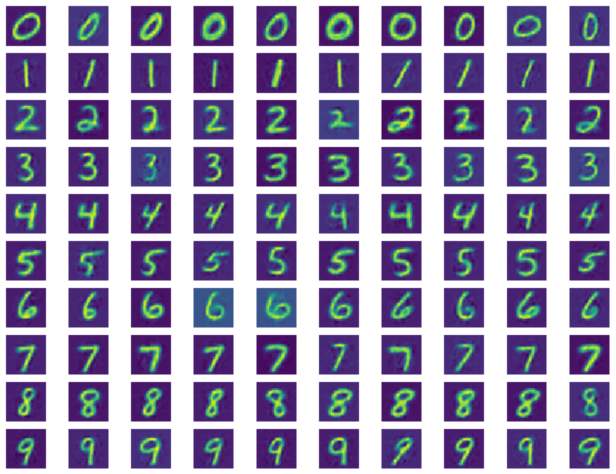


Figure 8: Images of the MNIST data. All of the images are produced by one densely connected neural network that has three hidden layers with 300 neurons each. We can generate images conditionally and here the network was asked to produce ten of each digit. We observe that the network can produce different characters and the variation within the character class. These figures are the results of epoch 250.

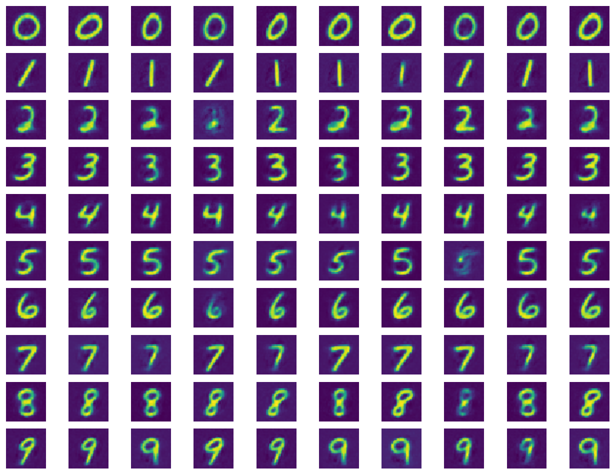
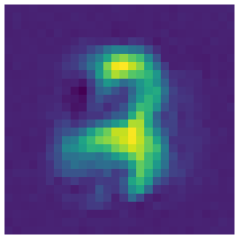


Figure 9: Images of the MNIST data. These figures have been produced with the correspondence finding batch size of 100 and the supervised learning batch size 100. The algorithm learns some variation in the data but to a lesser extent than with a bigger batch size.


(a) A character between 2 and 3. The character is supposed to be 2.

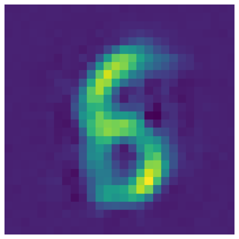

(b) A character between 5 and 6. The character is supposed to be 6.

Figure 10: In the early phase of the training (12th epoch) the neural net produces characters that seem to be mixes between two different categories. As the training progresses these mixes of two classes disappear.

4.3. Learning categorical distributions

We also tested how our algorithm would fit to learn categorical distributions. Figure 11 shows the result of one such distribution. Figure 12 shows a few convergence curves.

The distance measure that was used here was different from the cases seen previously. The neural network output was of the same dimension as the number of the categories in the distribution. We interpreted the maximum element of the prediction $\widehat{\mathbf{y}}$ to denote the selected class. The distance used here was the ”cross-entropy” of softmax:

$$d(\mathbf{y}, \widehat{\mathbf{y}}) = -\sum_i y_i \log \widetilde{y}_i \tag{2}$$

$$\widetilde{\mathbf{y}} = \mathrm{softmax}(\widehat{\mathbf{y}}). \tag{3}$$

Here $\mathbf{y}$ is a one-hot vector indicating the class.

5. Discussion

We have presented a simple algorithm to train a continuous mapping from a nearly arbitrary sampling distribution to a target distribution. The algorithm works for cases with a moderate amount of dimensions. It also necessitates that there is a meaningful distance for the distribution which we wish to approximate. This is not the case for example for natural images but there are for example reinforcement learning algorithms that could benefit from our algorithm (Rajamäki & Hämäläinen, 2017).

Because we are using supervised learning to train the mapping f, the algorithm is very robust, assuming that point-to-point correspondences are consistent between iterations, similarly to the stochastic minibatches in supervised training. Furthermore, because each data point is matched to a point in the input distribution, the algorithm cannot ignore any parts of the target distribution. Our simple synthetic example in Figure 4 demonstrates how the algorithm con-

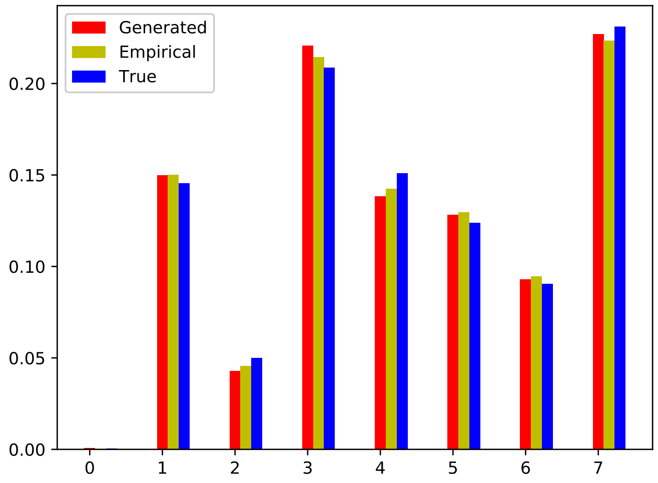


Figure 11: Mapping to a Multinoulli distribution with eight categories. The blue bars are true probability mass function values. The yellow bars denote the frequencies of 1000 data points sampled from the distribution and the red bars denote the frequencies of the generated data.

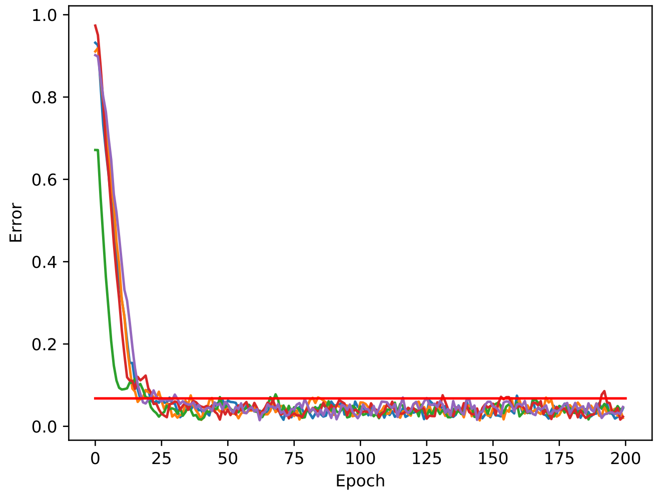


Figure 12: Convergence of the algorithm in the test of Figure 11. The error is the sum of the differences’ in the probability mass function. The curves are the errors of five different runs. The horizontal red line indicates the mean of errors between the true pmf and a 1000-sample empirical pmf.

verges even though there is minuscule overlap between the initially generated data and the true data.

5.1. Limitations

Each batch of data selected for the training procedure should be representative of the whole distribution being approximated. Likewise each batch of noise should be representative of the whole origin distribution. As seen in Section 4.2, the algorithm works with small minibatches, but as is the case with supervised learning, the quality is improved with a bigger batch size. The minibatch size of the supervised learning part of the algorithm can be chosen independently of the correspondence finding algorithm's batch size, though. Nonetheless, because of the quadratic computational cost of the correspondence finding algorithm, the balancing between a batch size that captures the variation in the data and a batch size that yields fast training becomes even more pronounced than with pure supervised learning.

The algorithm necessitates the existence of a meaningful distance in the space of the distribution being approximated. This is not the case for high-dimensional distributions such as distributions consisting of high-resolution images. Our algorithm works well with a moderate amount of dimensions, though, as demonstrated by the example with the MNIST data.

6. Conclusions

We have presented a robust and simple training algorithm to transform distributions in a moderate dimensional setting. The algorithm works well with and without conditioning variables and involves only one function approximator, such as a neural network. Be believe that there is still plenty of room for improvement in the algorithm, especially in developing correspondence procedures and distance metrics that produce consistent results with smaller minibatch sizes and high-dimensional data.

References

Arjovsky, Martin, Chintala, Soumith, and Bottou, Léon. Wasserstein Generative Adversarial Networks. In *International Conference on Machine Learning*, pp. 214–223, 2017.

Besl, Paul J and McKay, Neil D. A Method for Registration of 3-D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

Bojanowski, Piotr, Joulin, Armand, Lopez-Paz, David, and Szlam, Arthur. Optimizing the Latent Space of Generative Networks. *arXiv preprint arXiv:1707.05776*, 2017.

Bordes, Florian, Honari, Sina, and Vincent, Pascal. Learning to Generate Samples from Noise Through Infusion Training. In *International Conference on Learning Representations*, 2017.

Chen, Yang and Medioni, Gérard. Object Modeling by Registration of Multiple Range Images. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pp. 2724–2729. IEEE, 1991.

Dinh, Laurent, Krueger, David, and Bengio, Yoshua. NICE: Non-Linear Independent Components Estimation. *International Conference on Learning Representations*, 2015.

Dinh, Laurent, Sohl-Dickstein, Jascha, and Bengio, Samy. Density Estimation Using Real NVP. In *International Conference on Learning Representations*, 2016.

Eidnes, Lars and Nøkland, Arild. Shifting Mean Activation Towards Zero with Bipolar Activation Functions. *arXiv preprint arXiv:1709.04054*, 2017.

Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative Adversarial Nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Gulrajani, Ishaan, Ahmed, Faruk, Arjovsky, Martin, Dumoulin, Vincent, and Courville, Aaron. Improved Training of Wasserstein GANs. *arXiv preprint arXiv:1704.00028*, 2017.

Jang, Eric, Gu, Shixiang, and Poole, Ben. Categorical Reparameterization with Gumbel-Softmax. *International Conference on Learning Representations*, 2017.

Kingma, Diederik and Ba, Jimmy. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*, 2015.

Kingma, Diederik P and Welling, Max. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations*, 2014.

Klambauer, Günter, Unterthiner, Thomas, Mayr, Andreas, and Hochreiter, Sepp. Self-Normalizing Neural Networks. *arXiv preprint arXiv:1706.02515*, 2017.

Kohonen, Teuvo. Self-Organized Formation of Topologically Correct Feature Maps. *Biological Cybernetics*, 43 (1):59–69, 1982.

LeCun, Yann, Cortes, Corinna, and Burges, Christopher J.C. URL `http://yann.lecun.com/exdb/mnist/`.

Liu, Qiang and Wang, Dilin. Stein Variational Gradient Descent: A General Purpose Bayesian Inference Algorithm. In *Advances In Neural Information Processing Systems*, pp. 2378–2386, 2016.

Maddison, Chris J, Mnih, Andriy, and Teh, Yee Whye. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. *arXiv preprint arXiv:1611.00712*, 2016.

Rajamäki, Joose and Hämäläinen, Perttu. Augmenting Sampling Based Controllers with Machine Learning. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '17, pp. 11:1–11:9, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5091-4. doi: 10.1145/3099564.3099579. URL http://doi.acm.org/10.1145/3099564.3099579.