Householder Dense layer: an orthogonal weight parametrization for dimensionality reduction in Neural Networks

submitted in partial fulfillment for the degree of master of science

Jan Jetze Beitler
10416641

master information studies
data science
faculty of science
university of amsterdam

2018-06-26

	Supervisor
Title, Name	MSc, Ivan Sosnovik
Affiliation	UvA, FNWI
Email	sosnovikivan@gmail.com

Universiteit van Amsterdam

ABSTRACT

Orthogonal matrices have already been used in Neural Networks because of their good properties such as invertibility and norm preservation. However, orthogonal matrices can be used only in mappings directly parametrized by square matrices. This means that these mappings only allow for dimensionality preserving transformations. In order to achieve the desired properties with semi-orthogonal matrices, several techniques with use of Riemannian optimization over Stiefel manifolds were used. A disadvantage of this method is the requirement of complicated computations. Besides, such a mapping is non-invertible. Here we present a method that combines several useful properties of orthogonal matrices and allows for dimensionality reduction at the same time. We demonstrate that this method is able to achieve similar results to comparable methods. The method we propose, however, has significantly fewer parameters and it converges to be invertible.

1 INTRODUCTION

In many machine learning and computer vision tasks, such as text translation, object classification, image analysis, object tracking, etc. the problem can be formulated as an approximation of the relationship between an input (raw data) and a target. During the lasts decade, Deep Neural Networks demonstrated a great success in the above-described tasks. A Deep Neural Network (DNN) is a composition of multiple simple nonlinear functions. Each function usually consists of a linear transformation with learnable weights followed by a nonlinear element-wise operation.

By training, a DNN is able to represent a complex and nontrivial relationship between an input and an output. However, it is also prone to fitting trained data too well not being able to generalize for unseen data. Orthogonal matrices are able to solve this problem due to their ability to optimize over low embedded submanifolds [7].

Yet another great concern in machine learning is the problem of vanishing and exploding gradients in Recurrent Neural Networks (RNNs)[2] [18] [8] [12]. Especially when training long-term dependencies, gradients can become so small or large, that weights are updated in the same magnitude. A spectral norm smaller then one can raise exploding gradients, while a spectral norm bigger then one can cause vanishing gradients. Orthogonal matrices have the property to preserve norm and thus repeated iterative multiplication of a vector with an orthogonal matrix doesn't affect the norm of this vector. Because of this property, parameterizing a transition matrix in such a way that it is orthogonal, can thus solve the problem of vanishing or exploding gradients.

Orthogonal matrices are also used in normalizing flows [10] [16]. By applying a series of invertible functions to an input, a more flexible distribution can be obtained in the output. This flexibility decreases the distance between the derived and the true posterior distribution of the output.

However, in all of the above-mentioned cases multiplying a vector with the orthogonal matrix returns a vector with the same dimensions. This means that such matrices should be incorporated in Neural Networks along with mappings that do reduce the number of dimensions. In this case, the size of the Network and its learnable parameters does increase. This raises one question: Is it possible to create an orthogonal mapping that normalizes a vector, reduces the number of its dimensions and is invertible at the same time?

In current paper, we present a method that combines several useful properties of orthogonal matrices and allows for dimensionality reduction at the same time. We modify the general linear (fully-connected) layer, by replacing the arbitrary matrix with an orthogonal one. The output of the layer is split into two parts. The first one is feed to the next layer, while the second is penalized to be equal to zero. The mapping is absolutely invertible by design up until the splitting operation. The total function is trained to be invertible on the provided data samples.

The main contributions of our method, which will be shown in the results, are:

- The proposed method preserves the distributions with zero mean and unit variance up to a scale factor and a shift.
- The trained layer approximates an invertible nonsingular mapping.
- The proposed parametrization of the mapping is low-parametric. It stores significantly less number of trainable weights comparing to the widely-used fully-connected layer.

The current paper is structured as follows: the next section discusses the related work. Section 3 describes the main idea of the current method. We discuss the theoretical aspects of the mapping, as well as its effective implementation. The exhaustive set of experiments is demonstrated in Section 4. We summarize the paper and discuss the results in section 5 and 6.

2 RELATED WORK

The problem of vanishing and exploding gradients in RNNs could be solved by parameterizing the transition matrices with unitary weight matrices[2], which is analogue with orthogonal matrices but for the complex domain. Because of the norm preserving property of these matrices, the vanishing and exploding gradient problem is tackled. However, the parametrization used is not able to cover all unitary matrices if the weight matrix has more then 7 dimensions. By constraining the gradient to lie on the Stiefel manifold, a more complete set of unitary matrices can be constructed [18]. Yet another limitation of the parametrization used is the fact that it creates square matrices. This implies that multiplying a vector with this matrix returns a vector with the same dimensions. Such parametrization would increase the size of the DNNs as these Networks are dependent on reducing the number of dimensions. By creating rectangular orthogonal matrices, a DNN could be regularized with the use of orthogonal matrices[7].

Orthogonal matrices are also utilized in normalizing flows to enrich the variational posterior distribution. A normalizing flow is a series of invertible transformations. Utilizing an orthogonal matrix normalizes activations and offers the ability to de-correlate if the input is whitened. [16] utilizes this property by implementing a weight matrix constructed from a series of Householder matrices.

The matrix is placed in between two layers that reduce the number of dimensions of its input. Again, the orthogonal matrix itself does not reduce dimensions.

Invertibility of mappings can be achieved by using other types of transformations such as real-valued non-volume preserving (real NVP) transformations. Another method of normalizing activations was introduced by [3]. It allows for approximating more flexible distributions. Real NVP is able to perform efficient and exact inference. Because the transformations are invertible, real NVP is also able to reconstruct the original input.

Another DNN architecture where invertibility is implemented is in RevNet [4]. Residual Networks pushed state-of-the-art performance on image classification as networks grow both deeper and wider. This, however, also implies more memory usage as the activation of each layer should be stored. RevNet circumvents this bottleneck with invertible layers that can reconstruct the activations of its preceding layer. This is, however, limited to a hand-full of non-reversible layers, such as max-pooling. In *i*-RevNet, these layers are substituted by linear and invertible mappings that reduce the spatial resolution [9]. This makes *i*-RevNet fully invertible up until the final layer, which projects onto the classes. This paper also proves that losing information is not a necessary property of a Network to learn representations that generalize well on complicated data.

Despite the success of the above-described methods, none of them demonstrated an approach of creating an invertible mapping that has zero mean and unit variance and reduces the number of dimensions at the same time.

3 METHODOLOGY

The architecture of proposed method is visualized in figure 1. This architecture can be represented as

$$G = S \circ f,\ where\ f\ -\ invertible$$

or

$$S(f(x)) = \tilde{x}, r \tag{1}$$

The first part, $f(x)$, is a linear invertible mapping from an input to an output. Because $f(x)$ is parametrized with orthogonal matrices, it accommodates useful properties of orthogonal matrices such as norm preservation. Because of the orthogonality restriction, this part of the method is invertible by design. The second part of the method, $S(\cdot)$, performs a splitting operation on the output of $f(x)$. It is trained to be invertible by penalizing its norm to be zero. Because of the property of norm preservation, the sum of norms of $\tilde{x}$ and r is equal to the norm of x. If the norm of r is 0, the norm of $\tilde{x}$ is equal to the norm of x and thus all information is kept in the new dimensions.

3.1 Orthogonal invertible module

Here we propose a linear invertible module of the form

$$y = Wx + b \tag{2}$$

with W being any invertible matrix. One such type of invertible matrices are orthogonal matrices. This is the class of matrices that satisfy

$$X^T X = XX^T = I \tag{3}$$

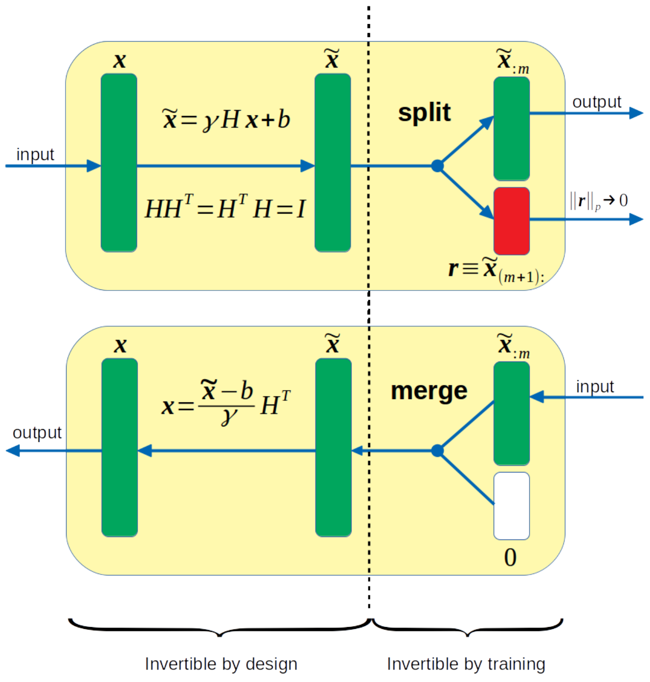


Figure 1: Architecture of proposed method. The first box describes the normal flow, the second box the inverse which is used in an autoencoder.

and therefore

$$Q^{-1} = Q^T \tag{4}$$

In current method, H is parametrized with a series of Householder transformations. The Householder transformation $x \longrightarrow Hx$ is defined by the matrix H of the following form [6]:

$$H = I - 2\frac{vv^T}{v^T v} \tag{5}$$

where H is orthogonal with determinant -1.

The Householder cascade is created by multiplying a series of Householder matrices each with a decreasing size of vector v. The size of this vector is determined by both n, the desired size of the cascade, and s, the step size each following vector should be decreased with. Each Householder matrix is placed in a subspace of an identity matrix with dimension n to allow for matrix multiplication.

$$W = H_n H_{n-1} \ldots H_2 \tag{6}$$

with

$$H_k = \begin{bmatrix} I_{n-k} & 0 \\ 0 & I_k - 2\frac{vv^T}{v^T v} \end{bmatrix} \tag{7}$$

As H is a reflection with a determinant of -1, W is either a reflection or a rotation, depending on the number of Householder matrices used in the cascade. This transformation is then scaled by γ and shifted by the bias-term to get the proposed form

$$\tilde{x} = \gamma Wx + b \tag{8}$$

The number of parameters for this method is dependent on both the desired dimension, n, of the Householder cascade and the step size, s, to decrease each factor with. The maximum number of parameters can be approximated by

$$\frac{n(n+s)}{2s} \tag{9}$$

3.2 Compact WY transformation

The naive Householder cascading implementation is a heavy computational algorithm as it needs to do many matrix multiplications. A more lightweight algorithm is based on the compact WY transformation and is able to reach the same outcome [14][21]. Here we refer to it as the Householder WY transformation.

The Householder WY transformation has the following form:

$$W = I_n - UT^{-1}U^T \tag{10}$$

with

$$U = \begin{bmatrix} v_n & v_{n-1} \dots v_2 \end{bmatrix},\ v_k = \begin{bmatrix} 0_{:n-k} \\ v \end{bmatrix}$$

and

$$T = \begin{bmatrix} 0.5 & & 1 \\ & \ddots & \\ 0 & & 0.5 \end{bmatrix} \times (U^T U)$$

Although the Householder WY transformation has fewer matrix multiplications to perform, it relies heavily on the computation of an inverse matrix. This implies that any gained efficiency is dependent on the machines ability to efficiently compute inverse matrices.

Figure 2 shows a comparison for the time consumption of both the naive Householder algorithm and the Householder WY transformation. The number of dimensions on the x-axis denotes the desired size of the cascade, vectors are created with a step size of 1. Time consumption is measured for both a single forward computation and a forward computation followed by backpropagation. For comparison reasons a third matrix was added which was initialized with random values. This figure shows that, especially for a growing number of dimensions, the Householder WY transformation is more efficient in terms of time consumption.

3.3 Splitting

The second part of the proposed method is reducing the number of dimensions. This is done by a function that splits the output of the mapping into a representation of the input distribution and a residual part.

$$S(f(x)) = [\tilde{x}_{:m}; r],\ r = \tilde{x}_{m:} \tag{11}$$

Because of the norm preserving property of orthogonal matrices, the output of $f(\cdot)$ has the same norm as its input. Penalizing the residual to be zero, drives its norm to zero. This implies that the norm of $\tilde{x}_{:m}$ tends to be equal to the norm of x. In other words, the information of the input distribution is transformed to the lower dimensional space of $\tilde{x}_{:m}$. By substituting the residual with zeros, the inverse of the mapping is able to transform the information from the low dimensional space into the original dimensional space. If $r \longrightarrow 0$ then

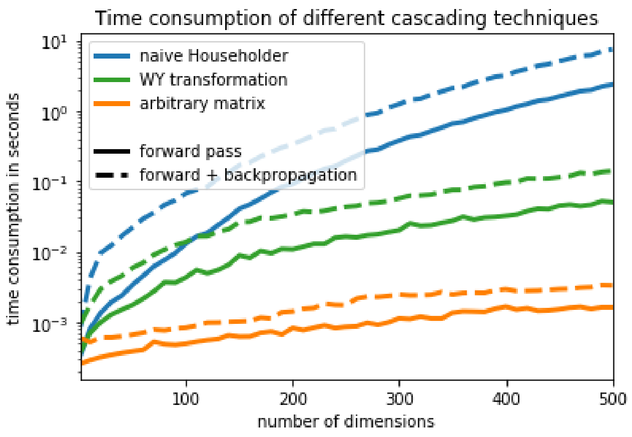


Figure 2: Comparison of the time consumption for different cascading techniques with a step size of 1 for both a single forward pass and a forward pass followed by backpropagation. The number of dimensions represents the number of columns, and thus number of rows, of the Householder cascade. Shown experiments are executed on a Tesla K80 GPU.

$$f^{-1}([\tilde{x}_{:m}; 0]) \longrightarrow x$$

and thus

$$||x - f^{-1}(f(x))||_p \longrightarrow 0 \tag{12}$$

The amount of information lost in the split is computed as the norm of r normalized to the norm of $f(x)$, $(\frac{||r||_2}{||\tilde{x}||_2})^2$.

In order to penalize r towards zero the norm of r is added to the Networks loss in form of a regularization term. To emphasize the importance of the fraction of information kept, the regularization term is multiplied with β. A higher β increase the regularization term which triggers the Network to put more effort in decreasing the norm of r. This way, less information is lost in the split.

$$L_{reg} = ||[\tilde{x}_{:m}, r] - [\tilde{x}_{:m}, 0]||_p = ||r||_p \tag{13}$$

$$L = L_0 + L_{reg} * \beta \tag{14}$$

However, when implementing multiple Householder Dense layers in an architecture, we want to minimize the loss of information of the input, not just the loss of information of the last layer. To calculate the total loss of information, we need to take the product of information kept in each layer and subtract this from one. The information kept in a layer is the information lost, subtracted from 1. The square root of this gives us the multiple layer loss. Let n be the number of Householder Dense layers, then the information lost is equal to

$$L_{reg} = 1 - \prod_{i=1}^{n} \left[1 - \left(\frac{||r^i||_2}{||\tilde{x}^i||_2} \right)^2 \right] \tag{15}$$

3.4 Normalization

Equation 16 proves that multiplying vector x with parametrized matrix H does not change the distribution of x. This implies that multiplication with H keeps Zero Mean and Unit Variance (ZMUV).

Method	Invertible	Preserves ZMUV	Allows dimensionality reduction	# trainable weights
Linear	−	−	+	NM
uRNN [18]	+	+	−	
HH-Flows [16]	+	+	−	
real NVP [3]	+	−	−	
OWN [7]	−	+	+	NM
Ours	±	+	+	$\frac{N(N+s)}{2s}$

Table 1: Comparison of properties. s is a hyper-parameter for the step size in choosing factors for the Householder Dense layer.

For this, assume the mean of x is $\mathbb{E}[x] = 0$ and the covariance matrix of x is $cov(x) = \sigma^2 I_{n\times n}$. Then

$$\begin{aligned} \mathbb{E}[Hx] &= \mathbb{E}[H]\mathbb{E}[x] = 0 \\ cov(Hx) &= \mathbb{E}[(Hx - \mathbb{E}[Hx])(Hx - \mathbb{E}[Hx])^T] \\ &= \mathbb{E}[Hx(Hx)^T] \\ &= \mathbb{E}[Hxx^T H^T] \\ &= H\mathbb{E}[xx^T]H^T \\ &= Hcov(x)H^T \\ &= H\sigma^2 I_{n\times n} H^T \\ &= \sigma^2 I_{n\times n} HH^T \\ &= \sigma^2 I_{n\times n} \end{aligned} \tag{16}$$

However, in proposed method vector x is not only multiplied with matrix H but also it is multiplied with γ and a bias is added. Equation 17 shows that, because of these steps, the standard deviation of input x gets multiplied by γ^2 and the mean of its distributions gets the bias added. This means that the distribution of vector x gets scaled and shifted.

$$\begin{aligned} \mathbb{E}(\gamma Hx + b) &= \mathbb{E}[\gamma]\mathbb{E}[Hx] + \mathbb{E}[b] = b \\ cov(\gamma Hx + b) &= \gamma^2 cov(Hx) \\ &= \gamma^2\sigma^2 I_{n\times n} \end{aligned} \tag{17}$$

The last process in proposed method is splitting the output vector. Equation 18 proves that this split operation does not alter the distribution.

$$\begin{aligned} \mathbb{E}[x_{:m}] &= \mathbb{E}[I_{m\times n}(\gamma Hx + b)] \\ &= I_{m\times n}\mathbb{E}[\gamma Hx + b] \\ &= I_{m\times n} b \\ &= b_{:m} \\ cov(x_{:m}) &= cov(I_{m\times n}(\gamma Hx + b)) \\ &= I_{m\times n} cov(\gamma Hx + b) I_{m\times n}^T \\ &= \gamma^2\sigma^2 I_{m\times m} \end{aligned} \tag{18}$$

This summarizes proposed method from equation 8 to have:

$$\begin{aligned} \mathbb{E}[S(f(x))] &= b_{:m} \\ cov(S(f(x))) &= \gamma^2\sigma^2 I_{m\times m} \end{aligned} \tag{19}$$

4 EXPERIMENTS

The experiments in this section are conducted on two image classification datasets, MNIST and FashionMNIST. Both datasets consist of 60.000 train and 10.000 test images of size 28 by 28. Each image has a single color channel and is labeled with one out of ten possible labels. The MNIST dataset consists of handwritten digits. As state-of-the-art architectures already outperformed human classification on MNIST with an accuracy of 99.79 percent [17], a more sophisticated dataset with the same accessibility, FashionMNIST, was provided [19]. The FashionMNIST dataset consists of 10 different product types from the Zalando website and has, to the best of our knowledge, a state-of-the-art accuracy of 94.41 percent [11] and a top-5 state-of-the-art accuracy of 89.35 percent [1]. Although both datasets do not represent real-life classification tasks they allow for low-computably tests on prototype algorithms. Because of the time restriction for this research, proposed algorithm was not tested on larger datasets. One exception was made for the experiment of normalization, which was conducted on CIFAR10, a dataset containing real-life images of size 32 by 32. Each image in this dataset consists of 3 color channels and is labeled with one out of ten labels.

The experiments described were all done with a step size, for the creation of the Householder cascade, of 10. This showed the best trade-off between different performance measures and the number of trainable parameters. For the penalization term the Frobenius norm was chosen as it gave the best results in a trade-off between accuracy and the loss of information. The presented method was, unless mentioned otherwise, implemented in a LeNet [20] architecture as replacement of the first fully-connected Dense layer. From now on we will refer to this architecture as HouseNet. Performance was then tested against a LeNet architecture with the original Dense layer and the same number of units. This architecture is shown in figure 3. In the remainder of this paper, we will call a layer parametrized with proposed method a Householder Dense layer.

Table 1 shows a comparison of different properties of our method with others. Both the columns for invertibility and Zero Mean Unit Variance will be shown in the following subsections.

4.1 Normalization

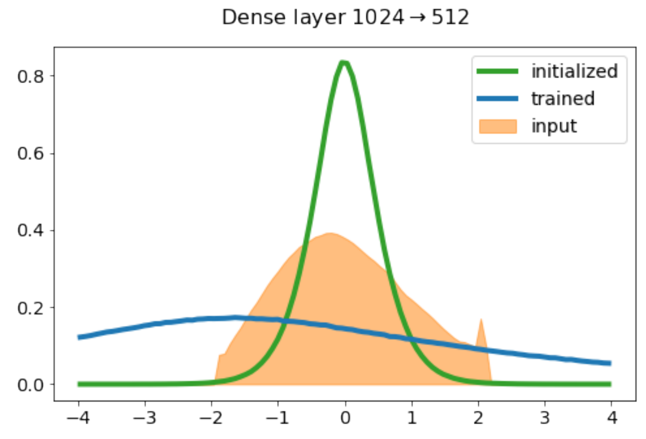
Figures 4a and 4b show the capability of both a Dense and a Householder Dense layer to normalize its output. The networks consist of two layers, both either Dense layers or Householder Dense layers. The red and green line represent the output distribution of the first layer.

Accuracy Comparison				
Model settings	Mnist	FashionMnist	Seconds per epoch	FC parameters
LeNet	0.9900	0.8903	**2.88**	32896
LeNet with OWN	0.9848	0.8772	77.33	32896
HouseNet, β 0.05	**0.9904**	**0.8921**	5.58	**3662**
HouseNet, β 0.1	0.9888	0.8807	5.58	**3662**

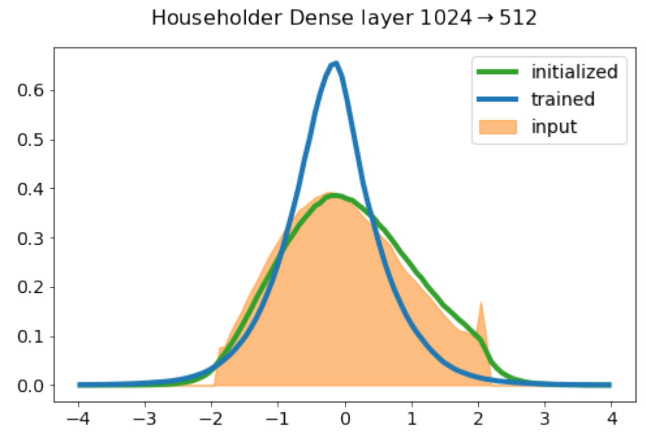
Table 2: Comparison of accuracies for different models. Experiments were conducted with a single linear layer with 128 units. Bold numbers are best scores.

$$\text{input} \xrightarrow[\text{6@3x3}]{\textbf{Conv layer}} \text{ReLU} \xrightarrow[\text{2x2}]{\textbf{MaxPool}} \text{ReLU} \xrightarrow[\text{16@5x5}]{\textbf{Conv layer}} \text{ReLU} \xrightarrow[\text{2x2}]{\textbf{MaxPool}} \text{ReLU} \xrightarrow[\text{128 units}]{\textbf{Dense layer}} \text{ReLU} \xrightarrow[\text{10 units}]{\textbf{Dense layer}} \text{x} \xrightarrow{\textbf{softmax}} \text{output}$$

Figure 3: The architecture of a basic LeNet model.



(a) Normalization of Dense layer.



(b) Normalization of Householder Dense layer.

Figure 4: Normalization on CIFAR10. Both models were made up of either two Householder Dense layers or two basic Dense layers. The red and green line represent the distribution of the first layer's output.

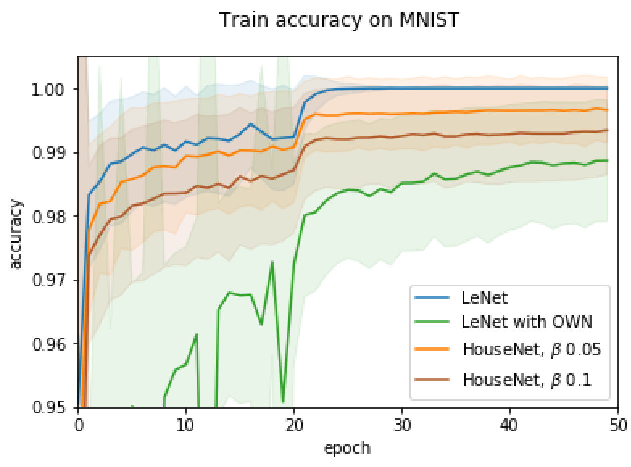
The orange area represents the distribution of CIFAR10, the input for the first layer in both models, which has a ZMUV distribution on initialization. As shown in figure 4b the Householder Dense layer follows the same ZMUV distribution. This reinforces equation 16 as the bias in this layer is initialized with a value of zero and γ with one. The Dense layer in figure 4a shows a small deviation from one in the variance, but it still follows a Gaussian distribution.

As the networks get input images to train on, the distributions change. Figure 4a shows that a Dense layer, after training, gets distributed within a wide range. As the next layer assumes the input distribution to be ZMUV it will get difficulties adapting its values to this widespread distribution. The Householder Dense layer, however, keeps a distribution that is quite similar to ZMUV. It got scaled by γ that was trained and is not equal to one anymore, and shifted by the bias. These findings reinforce the prove of equation 19 that proposed method preserves ZMUV up to a scale and shift.

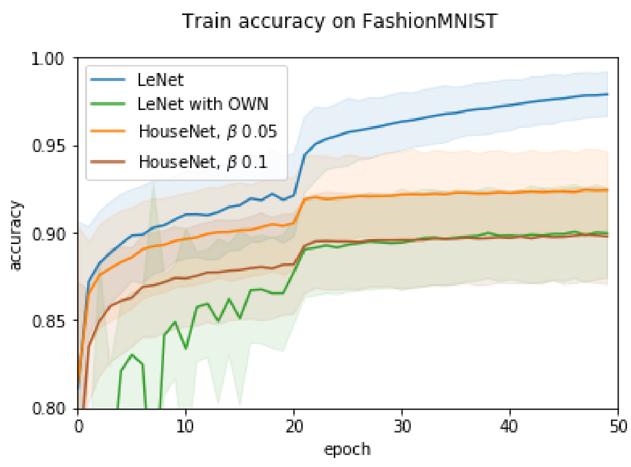
4.2 Orthogonal Linear model

To compare the performance of proposed method, we trained the basic LeNet model, LeNet with Orthogonal Weight Normalization (OWN) [7] and HouseNet, all with 128 units in the fully-connected layer. For the penalization of the Householder Dense layer two β's were chosen, 0.05 and 0.1, which showed the best results in a trade-off between high accuracy and minimum loss of information. Basic LeNet and HouseNet were trained for 50 epochs with Adam optimizer and ReLU activation. The learning rate was initialized with 0.01 and decreased, after 20 epochs, to 0.001. For LeNet with OWN, the number of epochs was increased to 100 and the learning rate was decreased at 40 epochs.

Table 2 shows the results of the experiment. For both MNIST and FashionMNIST HouseNet was able to get the highest accuracy, although basic LeNet got a comparable score at both datasets. However, the number of parameters for the Householder Dense layer is significantly smaller compared to that of a basic Dense layer or OWN. But although the Householder Dense layer has almost a fraction 10 fewer parameters compared to a basic Dense layer, basic LeNet does train faster then HouseNet. This is probably due to the inverse matrix in the Householder Dense layer.



(a) Train accuracy per epoch for the MNIST dataset.



(b) Train accuracy per epoch for the FashionMNIST dataset.

Figure 5: Train accuracy for both MNIST and FashionMNIST. Solid lines are the mean per epoch, the lighter color is its standard deviation. For LeNet with OWN the number of epochs on the x-axis have to be multiplied by 2.

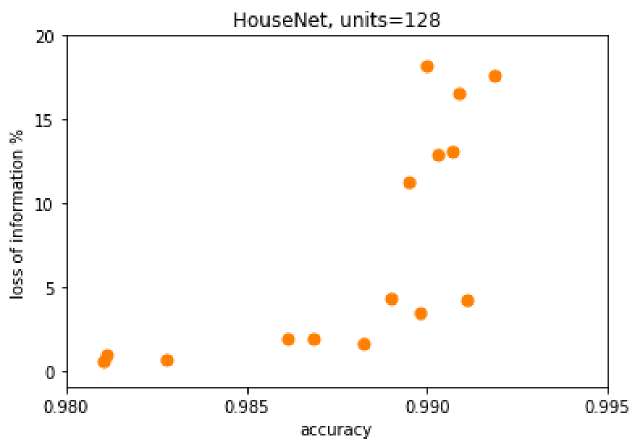


Figure 6: Trade-off between accuracy and loss of information. Plotted with several β's, low β's have higher loss of information.

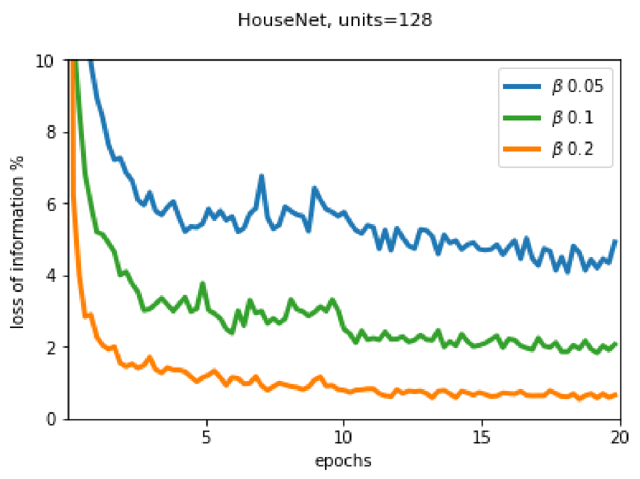


Figure 7: Loss of information per epoch on MNIST.

Figures 5a and 5b show the train accuracy per epoch. While both basic LeNet and HouseNet seem to be prone to overfitting, basic LeNet has the highest rate. This is especially visible on FashionMNIST where it almost reaches 99 percent accuracy on train data, but a ten percent point lower score on test data. For HouseNet, a higher beta seems to reduce the rate of overfitting. Compared to LeNet with OWN, both basic LeNet and HouseNet behave more stable during training. Something also pointed out by the paper about Orthogonal Weight Normalization[7]. However, the rate of overfitting is lower, especially on MNIST where the gap between train and test accuracy is less dan 0.005.

4.3 Loss of information

Several of the previous works showed that the loss of information may be a key for successful classification of images [15]. At the same time, it is not actually required because you can still classify correctly with no loss up until the last layer [9]. In figure 6 we show the continues dependency between the loss of information and the accuracy, which fits fine with the previous results. It is true that the loss is not actually required for successful results. Nevertheless, a minimal loss of info (5 percent) makes it easier for the neural network to achieve higher accuracy.

Figure 7 shows how the loss of information on train data evolves over time. Interestingly, after only five epochs the model already got to its stabilizing rate of information loss. At the tenth epoch there is another drop, because of reducing the learning rate from 0.01 to 0.001, but again the loss of information does not decrease significantly anymore. These remarks seem to especially hold for lower values of β.

$$\text{input} \xrightarrow[\text{717 units}]{\textbf{HH Dense}} \text{LeakyReLU} \xrightarrow[\text{350 units}]{\textbf{HH Dense}} \text{LeakyReLU} \xrightarrow[\text{250 units}]{\textbf{HH Dense}} \text{x} \xrightarrow[\text{350 units}]{\textbf{HH } \boldsymbol{i}\textbf{-Dense}} i\text{-LeakyReLU} \xrightarrow[\text{717 units}]{\textbf{HH } \boldsymbol{i}\textbf{-Dense}} i\text{-LeakyReLU} \xrightarrow[\text{784 units}]{\textbf{HH } \boldsymbol{i}\textbf{-Dense}} \text{output}$$

Figure 8: Architecture of autoencoder with HH *i*-Dense being the inverse of Householder Dense layer and *i*-LeakyReLU(x, α) defined as $min(x, \frac{x}{\alpha})$.

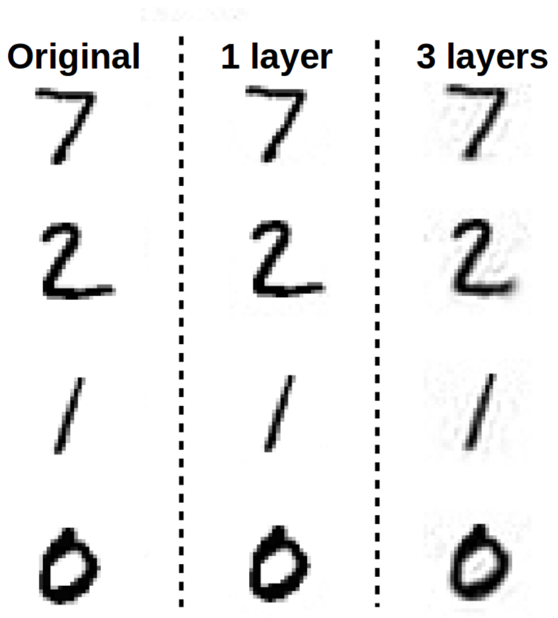


Figure 9: Output of the autoencoder. The single layer autoencoder reduced the number of dimensions from 784 to 717. The three layer autoencoder reduced the number of dimensions from 784 to 250.

4.4 Autoencoding

As proposed method is partially invertible by design and partially invertible by training, an autoencoder made up of only Householder Dense layers only has to train the encoding part of the Network. Once the dimensions are reduced to a certain number, the model can reproduce the original input by simply using the derived weights. For every layer first the residuals of the output vector are substituted with zeros, then the bias is subtracted and the outcome is divided by γ. At last, the vector is multiplied by the inverse of the trained transition matrix. This process can be even more simplified by the property of orthogonal matrices that its inverse is equal to its transpose. Retrieving the transposed of a matrix is much less computational intensive as retrieving its inverse is. The lower box in figure 1 visualizes this process of decoding.

In the autoencoder, however, an invertible activation function is crucial. As ReLU [13] is defined as $max(x, 0)$, no information is kept when an input is negative. LeakyReLU [5] is a modification of ReLU and is defined as $max(x, \alpha x)$ with a standard value for α being 0.2. With this activation function, the magnitude of each negative input is just reduced, each input value still having a unique output value. The inverse of LeakyReLU would be defined as $min(x, \frac{x}{\alpha})$.

For an autoencoder, there is no need to add a regularization term and if added, it behaves differently as in the case of classification. If X is the input and $\tilde{X}$ the output of an autoencoder, then

$$L_0 = \frac{1}{N}\sum_{i=0}^{N}(X_i - \tilde{X}_i)^2 \tag{20}$$

Because $\tilde{X} = f^{-1}([\tilde{x}_{:m}; 0])$ with 0 being the substitute of r, if $||r||_p \neq 0$ then $L_0 \neq 0$. This way, the deviation of r from zero is already incorporated into the loss function of an autoencoder. In this case, adding $||r||_p$ to the loss in the form of a regularization term merely serves as an emphasis of the importance of information preservation.

Summing pixels with the same index over all images in the train set of the MNIST dataset shows that 67 pixels do not have any information in any picture. This implies that 67 dimensions could be removed without any loss of information. This is shown in the second column of figure 9. These results were obtained with a single layer autoencoder with 717 units. After training for only two epochs a Mean Squared Error (MSE) and information loss rate of 0 were reached.

The first layer in our autoencoder thus reduces the number of dimensions from 784 to 717. The number of units in every next layer is determined to keep the loss of information in five epochs below one percent. This brought us to the next layers reducing the number of dimensions to 350 and 250. The architecture is shown in figure 8. The result is shown in the third column of figure 9. After training for 20 epochs, each with an average time of 34 seconds, the MSE was 0.0057 and the loss of information rate 0.0087. This shows that a series of blocks, consisting of a Householder Dense layer and LeakyReLU, is able to reduce the dimensions of the input data by almost 70 percent with a loss of information of less than one percent.

5 CONCLUSION

Orthogonal matrices are commonly used in different types of Neural Networks. In Recurrent Neural Networks, they are utilized because of their norm preserving property. By using orthogonal matrices the problem of vanishing and exploding gradients can be solved. Also in normalizing flows orthogonal matrices are used to obtain a more flexible distribution. In many of these applications, however, these matrices are not able to reduce the number of dimensions.

In this research, we use orthogonal matrices to create a normalizing, invertible mapping that is able to reduce the number of dimensions. We parametrize a weight matrix with a series of Householder matrices, which are orthogonal. By multiplying an input with the weight matrix and penalizing part of the output to be zero, the dimensions are reduced with a minimal loss of information. The layer constructed from this weight matrix, the Householder Dense layer, is able to reach similar performance, in terms of accuracy, compared to a basic Dense layer. The number of parameters is, however, reduced by almost a factor ten.

The research question to be answered was if it is possible to create a mapping that normalizes a vector, reduces the number of dimensions and is invertible at the same time. Table 1 in section 4 summarizes the answers to this question. In section 4.2 we showed that proposed method of reducing dimensions does indeed work and is able to reach comparable results with significantly fewer parameters. This was again shown in section 4.4 where an autoencoder of only Householder Dense layers was able to reduce the number of dimensions by almost 70 percent. At the same time, this section showed that the proposed mapping is invertible as it is able to reconstruct the original image from the lower dimensional space with only a minimal error. However, the method is partially invertible by design and partially by training. The ability to normalize a vector is given in section 4.1. Here it was shown that the method normalizes an input vector close to a Zero Mean Unit Variance (ZMUV) Gaussian distribution with a scale and shift. Although it does not normalize to exact unit variance, it performs much better compared to a basic Dense layer.

6 DISCUSSION

The experiments conducted in this research were all done, with exception of the test for normalization, on the MNIST and Fashion-MNIST datasets. As these are well-known toy-datasets, it could be interesting do mimic the experiments on more advanced datasets and test whether the conclusions still hold.

In this research, the experiments were conducted with a predefined set of hyper-parameters. Some of these hyper-parameters are the norm used in the penalization term, the factors with which the Householder cascade is created and the β used for penalization. It could be interesting to more intensively study their behavior as the chosen set might be just a sub-optimum. Also β is difficult to choose as the trigger for the Network to minimize the loss of information is also dependent on the magnitude of the basic loss.

ACKNOWLEDGMENTS

The writer of this research would like to take this moment to thank his supervisor, Ivan Sosnovik, for the great support during this research and 1-on-1 lectures he gave. Also he would like to thank his parents for their great support not only for the period of this research, but for as long as he has been studying.

REFERENCES

[1] Abien Fred Agarap. 2018. Deep Learning using Rectified Linear Units (ReLU). 1 (2018). http://arxiv.org/abs/1803.08375
[2] Martin Arjovsky, Amar Shah, and Yoshua Bengio. 2015. Unitary Evolution Recurrent Neural Networks. 48 (2015). http://arxiv.org/abs/1511.06464
[3] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. 2017. Density estimation using real NVP. (2017).
[4] Aidan N. Gomez, Mengye Ren, Raquel Urtasun, and Roger B. Grosse. 2017. The Reversible Residual Network: Backpropagation Without Storing Activations. (2017), 1–14. http://arxiv.org/abs/1707.04585
[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Proceedings of the IEEE International Conference on Computer Vision* 2015 Inter (2015), 1026–1034. DOI:http://dx.doi.org/10.1109/ICCV.2015.123
[6] Alston S. Householder. 1958. Unitary Triangularization of a Nonsymmetric Matrix. *J. ACM* 5, 4 (1958), 339–342. DOI:http://dx.doi.org/10.1145/320941.320947
[7] Lei Huang, Xianglong Liu, Bo Lang, Adams Wei Yu, Yongliang Wang, and Bo Li. 2017. Orthogonal Weight Normalization: Solution to Optimization over Multiple Dependent Stiefel Manifolds in Deep Neural Networks. (2017). http://arxiv.org/abs/1709.06079
[8] Stephanie L. Hyland and Gunnar Rätsch. 2016. Learning Unitary Operators with Help From u(n). 2 (2016), 2050–2058. http://arxiv.org/abs/1607.04903
[9] Jörn-Henrik Jacobsen, Arnold Smeulders, and Edouard Oyallon. 2018. i-RevNet: Deep Invertible Networks. (2018), 1–11. DOI:http://dx.doi.org/10.1051/0004-6361/201527329
[10] Danilo Jimenez Rezende and Shakir Mohamed. 2015. Variational Inference with Normalizing Flows. (2015). https://arxiv.org/pdf/1505.05770.pdf
[11] Franco Manessi and Alessandro Rozza. 2018. Learning Combinations of Activation Functions. (2018). http://arxiv.org/abs/1801.09403
[12] Zakaria Mhammedi, Andrew Hellicar, Ashfaqur Rahman, and James Bailey. 2016. Efficient Orthogonal Parametrisation of Recurrent Neural Networks Using Householder Reflections. (2016). http://arxiv.org/abs/1612.00188
[13] Vinod Nair and Geoffrey E Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of the 27th International Conference on Machine Learning* 3 (2010), 807–814. DOI:http://dx.doi.org/10.1.1.165.6419
[14] Robert Schreiber and Charles Van Loan. 1989. A Storage-Efficient WY Representation for Products of Householder Transformations. *SIAM J. Sci. Statist. Comput.* 10, 1 (1989), 53–57. DOI:http://dx.doi.org/10.1137/0910005
[15] Naftali Tishby and Noga Zaslavsky. 2015. Deep Learning and the Information Bottleneck Principle. (2015). DOI:http://dx.doi.org/10.1109/ITW.2015.7133169
[16] Jakub M Tomczak and Max Welling. 2017. Improving Variational Auto-Encoders using Householder Flow. (2017). https://arxiv.org/pdf/1611.09630.pdf
[17] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Lecun, and Rob Fergus. 2013. Regularization of Neural Networks using DropConnect. (2013). https://cs.nyu.edu/~wanli/dropc/dropc.pdf
[18] Scott Wisdom, Thomas Powers, John R. Hershey, Jonathan Le Roux, and Les Atlas. 2016. Full-Capacity Unitary Recurrent Neural Networks. Nips (2016), 1–9. http://arxiv.org/abs/1611.00035
[19] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. (2017), 1–6. http://arxiv.org/abs/1708.07747
[20] Y. LeCun, L. D.Jackel, L. Bottou, A. Brunot, C. Cortes, J.˜S.˜Denker, H.˜Drucker, I. Guyon, U. A. Müller, E. Säckinger, P. Simard, and V. Vapnik. 1995. Comparison of learning algorithms for handwritten digit recognition. *Proceedings {ICANN'95} - -International Conference on Artificial Neural Networks* II (1995), 53–60.
[21] Yusaku Yamamoto and Yusuke Hirota. 2011. A parallel algorithm for incremental orthogonalization based on the compact WY representation. *JSIAM Letters* 3, 0 (2011), 89–92. DOI:http://dx.doi.org/10.14495/jsiaml.3.89