MinvNet: Building Invertible Neural Networks with Masked Convolutions

Abstract

We show that invertible neural networks can be constructed using masked convolutions, with architectures similar to ResNets. Their inverses can be easily obtained by sequentially inverting univariate monotonic functions. The determinants of Jacobians can be computed analytically, enabling generative modeling based on the change of variables formula. We empirically demonstrate that these invertible neural networks can perform competitively with ResNets on classifying CIFAR-10 images. When trained as generative models, our invertible neural networks are able to set a new state-of-the-art record of likelihood on MNIST and match the state-of-the-art likelihood on CIFAR-10.

1. Introduction

Invertible neural networks provide a unique tool to understanding and tackling problems in classification and generative modeling. They have been employed to investigate representations of deep classifiers (Jacobsen et al., 2018), create generative models that are directly trainable by maximum likelihood (Dinh et al., 2016; 2015; Papamakarios et al., 2017; Kingma & Dhariwal, 2018; Grathwohl et al., 2018; Behrmann et al., 2019), and do approximate inference (Rezende & Mohamed, 2015; Kingma et al., 2016).

There are two important desiderata for invertible neural networks. First, there should exist efficient algorithms for inverting the network. Second, the determinant of the Jacobian should be tractable to enable applications in generative models. Many methods have been proposed to construct models that are both invertible and have tractable determinants of Jacobians. For example, planar flows (Rezende & Mohamed, 2015) and Sylvester flows (Berg et al., 2018) all have tractable determinants of Jacobians by design, but they have bottlenecks in the architecture that limit the largest dimension of hidden representations. NICE (Dinh et al., 2015), realNVP (Dinh et al., 2016) and Glow (Kingma & Dhariwal, 2018) also have tractable Jacobians. However, they rely on fixed dimension splitting heuristics and unorthodox architectures such as the coupling layers, which could make training and tuning harder. Methods like FFJORD (Grathwohl et al., 2018), i-ResNets (Behrmann et al., 2019) have less constrains on the architectures. Unfortunately, their Jacobian determinants can only be approximated.

In this paper, we propose a new invertible network based on masked convolutions. This new family of models, named *MinvNet*s, can be efficiently inverted by sequentially solving univariate equations, and their Jacobian determinants can be analytically computed. The architecture of MinvNet is similar to that of ResNet—the state-of-the-art architecture of discriminative learning, which arguably facilitates the learning of expressive features and deep models. Empirically, we found that a MinvNet classifier achieves 90.2% accuracy on CIFAR-10, which is comparable to the 92.6% accuracy achieved by a ResNet with a similar architecture. When using MinvNets as generative models, it achieves a bit per dimension (bpd) of 0.98 on MNIST, outperforming the former state-of-the-art (0.99 from FFJORD (Grathwohl et al., 2018)). On CIFAR-10, the bpd of MinvNet is 3.35, which is the same as state-of-the-art result achieved by Glow (Kingma & Dhariwal, 2018).

2. Background

2.1. Understanding classification with invertible neural networks

Enforcing the invertibility of a neural network leads to more interpretable classifiers. Before invertible neural networks were invented, in order to understand what input leads to a specific label, prior work (Dosovitskiy & Brox, 2016; Mahendran & Vedaldi, 2016) inverted the representations by means of learned or hand-engineered priors. For invertible neural networks, it is easy to trace down a prediction by inverting the final representations. This has been shown useful for understanding distributions of latent features (Jacobsen et al., 2018) and analyzing the cause of adversarial examples (Jacobsen et al., 2019).

2.2. Generative modeling with invertible neural networks

An invertible neural network $f : \mathbf{x} \in \mathbb{R}^n \mapsto \mathbf{z} \in \mathbb{R}^n$ can be used to map a usually complex probability density $p(\mathbf{x})$ to a simple base distribution $\pi(\mathbf{z})$. The change of variable formula relates the densities before and after the

transformation, *i.e.*,

$$p(\mathbf{x}) = \pi(f(\mathbf{x})) \log \left| \det \left(\frac{\partial f}{\partial \mathbf{x}} \right) \right|, \tag{1}$$

where $\frac{\partial f}{\partial \mathbf{x}}$ denotes the Jacobian of $f(\mathbf{x})$. If the Jacobian determinant $\det(\frac{\partial f}{\partial \mathbf{x}})$ is tractable, $p(\mathbf{x})$ can be easily computed given $f(\mathbf{x})$, using the above formula. Therefore $f(\mathbf{x})$ can be used to represent a probability density $p(\mathbf{x})$. In addition, samples from $p(\mathbf{x})$ can be obtained by first drawing $\mathbf{z} \sim \pi(\mathbf{z})$, and then computing the inverse $f^{-1}(\mathbf{z})$.

3. Constructing Invertible Layers with Masked Convolutions

3.1. The basic invertible module

We start from considering linear transformations $f(\mathbf{x}) = W\mathbf{x} + \mathbf{b}$, $W \in \mathbb{R}^{n \times n}$, $\mathbf{b} \in \mathbb{R}^n$. Obviously f is invertible as long as W is not singular. However, for a general W, computing its inverse and determinant requires $O(n^3)$ operations. For more efficient inversion and determinant computation, we choose W to be a triangular matrix with nonzero diagonal entries. The function $f(\mathbf{x}) = W\mathbf{x} + \mathbf{b}$ is our basic invertible module.

Because W is triangular, $f(\mathbf{x})$ can be efficiently inverted with forward / backward substitution, which only costs $O(n^2)$ computations. For example, suppose W is lower triangular and consider computing $\mathbf{x} = f^{-1}(\mathbf{y})$, *i.e.*, solving the following linear system

$$\begin{array}{ccccccccc} W_{11}x_1 & & & & & & & = & y_1 - b_1 \\ W_{21}x_1 & + & W_{22}x_2 & & & & & = & y_2 - b_2 \\ \vdots & & \vdots & & \ddots & & & & \vdots \\ W_{n1}x_1 & + & W_{n2}x_2 & + & \cdots & + & W_{nn}x_n & = & y_n - b_n \end{array}.$$

The solution for $x_1, x_2, \cdots, x_n$ can be obtained sequentially. We first solve for x_1 using the first equation $W_{11}x_1 + b_1 = y_1$. The second equation only involves x_1 and x_2, so x_2 can be solved because we already know x_1. Continuing in this way, x_k can be solved given the values of $x_1, x_2, \cdots, x_{k-1}$.

Moreover, the determinant of Jacobian for $f(\mathbf{x})$ only requires $O(n)$ computations, since the Jacobian is W, and the determinant of a triangular matrix is the product of its diagonal entries.

3.2. The calculus of invertible modules

More complex and expressive invertible functions can be constructed from our basic invertible module. Without loss of generality, we assume the weight matrix W of our basic block is lower triangular. We will consider three composition rules as our tools of building more expressive invertible functions. These composition rules preserve the property of triangular Jacobians so that the Jacobian determinants of the composed modules are easy to compute. We will next discuss additional constraints needed to make the composed modules invertible.

We already know that $f(\mathbf{x}) = W\mathbf{x}+\mathbf{b}$ has a lower triangular Jacobian. New modules with lower triangular Jacobians can be created with the following composition rules.

- **Addition**. If f_1 and f_2 have lower triangular Jacobians, we can build a new module by $f = f_1 + f_2$. The Jacobian of f satisfies $\frac{\partial f}{\partial \mathbf{x}} = \frac{\partial f_1}{\partial \mathbf{x}} + \frac{\partial f_2}{\partial \mathbf{x}}$, and is therefore lower triangular.
- **Non-linearity**. If f_0 has a lower triangular Jacobian, a new module can be created by $f(\mathbf{x}) = h(f_0(\mathbf{x}))$, where $h(\cdot)$ is an elementwise activation function that transforms the output of f_0. The Jacobian of f is still lower triangular, because $\frac{\partial f}{\partial \mathbf{x}} = \mathrm{diag}(h'(f_0(\mathbf{x}))) \frac{\partial f_0}{\partial \mathbf{x}}$. Here $\mathrm{diag}(\mathbf{v})$ represents a diagonal matrix with $\mathbf{v}$ on the diagonal.
- **Composition**. If f_1 and f_2 have lower triangular Jacobians, we can create a new module by $f = f_2 \circ f_1$. The Jacobian of f is $\frac{\partial f}{\partial \mathbf{x}} = \frac{\partial f_2}{\partial \mathbf{x}}\big|_{\mathbf{x}=f_1(\mathbf{x})} \frac{\partial f_1}{\partial \mathbf{x}}$, and is also lower triangular.

Next, we state the sufficient condition for a module with a lower triangular Jacobian to be invertible. Let the function be $f : \mathbb{R}^n \to \mathbb{R}^n$, and denote the Jacobian of f at $\mathbf{x}$ as $J_f(\mathbf{x})$. We use $\mathrm{diag}(J_f(\mathbf{x}))$ to denote the diagonal of $J_f(\mathbf{x})$, and all inequalities are considered elementwise. The condition for f being invertible is summarized as the following theorem.

Theorem 1. *If the following conditions hold for* $\forall \mathbf{x} \in \mathbb{R}^n$*: (i)* $J_f(\mathbf{x})$ *is lower triangular; (ii)* $\mathrm{diag}(J_f(\mathbf{x})J_f(0)) > 0$*, then* f *is invertible. Moreover,* $\mathbf{x} = f^{-1}(\mathbf{y})$ *can be obtained by sequentially inverting* n *univariate monotonic functions.*

Proof. See the appendix. □

In other words, the new modules created by the rules of addition, non-linearity or composition will be invertible as long as each diagonal entry of their Jacobians has the same sign across all the inputs. When this condition is satisfied, the inverse can be obtained efficiently using a procedure similar to forward substitution of the basic invertible module. Note that univariate monotonic functions can be inverted efficiently, using—for example—the Newton-Raphson method.

3.3. The invertible layer

Using the basic blocks and composition rules, we can construct an expressive invertible module whose inver-

sion and determinant of Jacobian can be efficiently computed. The invertible layer includes 3 groups of basic invertible modules on $\mathbb{R}^n$, whose weights are $\{(W_i^1, \mathbf{b}_i^1)\}|_{i=1}^k$, $\{(W_{ij}^2, \mathbf{b}_{ij}^2)\}|_{1\le i,j\le k}$, and $\{(W_i^3, \mathbf{b}_i^3)\}|_{i=1}^k$. In addition, the invertible layer also uses an activation function h, which we assume to be strictly increasing. All matrices involved are lower triangular.

We define our layer as

$$\ell(\mathbf{x}) \triangleq \sum_{i=1}^{k} W_i^3 h\Big(\sum_{j=1}^{k} W_{ij}^2 h(W_j^1\mathbf{x}+\mathbf{b}_j^1)+\mathbf{b}_{ij}^2\Big)+\mathbf{b}_i^3,$$

which resembles the form of a 3-layer neural network with k hidden units. By the composition rules in Section 3.2, $\ell(\mathbf{x})$ has a lower triangular Jacobian. The analytic form of the Jacobian is

$$J_\ell(\mathbf{x}) = \sum_{i=1}^{k} W_i^3 \alpha_i \sum_{j=1}^{k} W_{ij}^2 \beta_j W_j^1,$$

where $\alpha_i = h'\big(\sum_{j=1}^k W_{ij}^2 h(W_j^1\mathbf{x}+\mathbf{b}_j^1)+\mathbf{b}_{ij}^2\big)$ and $\beta_j = h'(W_j^1\mathbf{x}+\mathbf{b}_j^1)$. Note that α_i and β_j are all positive because h is strictly increasing and $h' > 0$. Therefore, the condition of Theorem 1 can be satisfied when

$$\operatorname{diag}(W_i^3)\operatorname{diag}(W_{ij}^2)\operatorname{diag}(W_j^1) > 0, \forall 1 \le i,j \le k, \quad (2)$$

in which case $\operatorname{diag}(J_\ell(\mathbf{x})) > 0$ is guaranteed for all $\mathbf{x} \in \mathbb{R}^n$. Since Eq. (2) only involves the diagonal terms of weight matrices, it only affects $\frac{2}{n+1}$ of all variables and is arguably negligible when n is large.

As a result, when the constraints of diagonals in Eq. (2) are satisfied, $\ell(\mathbf{x})$ is an invertible module with a lower triangular Jacobian. From Theorem 1, the inverse of $\ell(\mathbf{x})$ can be computed by solving n monotonic univariate functions sequentially.

We next modify $\ell(\mathbf{x})$ to follow the form of a residual block:

$$r(\mathbf{x}) \triangleq \mathbf{t} \odot \mathbf{x} + \ell(\mathbf{x}), \quad (3)$$

where $\mathbf{t} > 0$ and $\odot$ is element-wise multiplication. Following the same reasoning, $r(\mathbf{x})$ has a lower triangular Jacobian, and is invertible under the same constraints of Eq. (2). In practice, this residual block performs better when a large number of invertible layers are chained together.

3.4. Masked convolutions

Convolution is a special type of linear transformation. We note that with appropriate masks, convolutions correspond to triangular matrices (*cf*., causal convolutions used in Oord et al. (2016)). In our experiments, we use masked convolutions to implement the invertible layer discussed in Section 3.3 for image classification and density estimation. Suppose the input to our invertible layer is $\mathbf{x} \in \mathbb{R}^{C\times H\times W}$, where C denotes the number of channels, and H, W denote height and width respectively. Let $M \in \mathbb{R}^{C_1\times C_2\times R\times R}$ denote a masked convolutional filter that has C_1 input channels, C_2 output channels, and a filter size of $R \times R$. We need 3 masked convolutions $M^1 \in \mathbb{R}^{C\times kC\times R\times R}$, $M^2 \in \mathbb{R}^{kC\times kC\times R\times R}$ and $M^3 \in \mathbb{R}^{kC\times C\times R\times R}$ for the 3 groups of basic invertible modules respectively. The residual version of the invertible layer can be written as

$$r(\mathbf{x}) = \mathbf{t}\odot\mathbf{x} + M^3 \circledast h(M^2 \circledast h(M^1 \circledast \mathbf{x}+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3, \quad (4)$$

where $\circledast$ denotes discrete convolution. The filters of M^1, M^2, M^3 are all constrained such that Eq. (2) holds true. However, this constraint only affects $\frac{2}{R^2}$ of all entries in the filters, since Eq. (2) only involves diagonals.

4. Building Invertible Neural Networks

In this section, we discuss how to combine multiple invertible layers together to form a deep invertible neural network, which we call *MinvNet*. We focus on the invertible layer with residual connections and masked convolutions (Eq. (4)).

Grouped invertible layers Previously, we often assume the basic invertible module uses a lower triangular matrix to simplify the discussion, and the invertible layer has a lower triangular Jacobian. To maximize the expressive power of our invertible neural network, it is undesirable to constrain the Jacobian of the network to be triangular. We thus always group two invertible layers together as a chain—one has a lower triangular Jacobian and the other has an upper triangular Jacobian. The grouped layers are still invertible since each layer in the group is invertible. The Jacobian of the grouped layers are no longer triangular, but its determinant is still easy to compute since it is the product of the determinants of two triangular matrices.

Invertible subsampling Subsampling is important for enlarging the receptive field of convolutions. However, common subsampling operations such as pooling and strided convolutions are usually not invertible. Following Dinh et al. (2016) and Behrmann et al. (2019), we use a "squeezing" operation to reshape the feature maps so that they have smaller resolution but more channels. After a squeezing operation, the height and width will decrease by a factor of 2, but the number of channels will increase by a factor of 4. This procedure is invertible and the Jacobian is identity.

5. Experiments

In this section, we test our MinvNet on both image classification and density estimation. We focus on two common

image datsets, MNIST and CIFAR-10.

5.1. Classification

To check the expressive power of MinvNet, we test its classification performance on CIFAR-10, and compare it to a ResNet with a similar architecture.

Architecture The ResNet contains 38 pre-activation residual blocks (He et al., 2016), and each block has three 3×3 convolutions. The architecture is divided into 3 stages, with 16, 64 and 256 filters respectively. Our MinvNet uses 19 grouped invertible layers, which include a total of 38 residual invertible layers, each having three 3×3 convolutions. Batch normalization is applied before each invertible layer. Note that batch normalization does not affect the invertibility of our network, because during test time it uses fixed running average and standard deviation and is an invertible operation. We use 2 squeezing blocks at the same position where ResNet applies subsampling, and matches the number of filters used in ResNet. To produce the logits for classification, both MinvNet and ResNet first apply global average pooling and then use a fully connected layer.

Setup Following Behrmann et al. (2019), we pad the images to 16 channels with zeros. This corresponds to the first convolution in ResNet which increases the number of channels to 16. Both ResNet and our MinvNet are trained with AMSGrad (Reddi et al., 2018) for 300 epochs with the cosine learning rate schedule (Loshchilov & Hutter, 2016) and an initial learning rate of 0.001. Both networks use a batch size of 128.

Results On the test dataset, MinvNet achieves a test accuracy of 90.2% while ResNet achieves 92.6%. Both MinvNet and ResNet achieves 100% accuracy on the training dataset. This indicates that MinvNet has enough representation power to fit all data labels on the training dataset, and the invertible representations learned by it are comparable to representations learned by non-invertible networks.

5.2. Density Estimation

Setup We mostly follow the settings in Papamakarios et al. (2017). All training images are dequantized and transformed using the logit transformation. Networks are trained using AMSGrad (Phuong & Phong, 2019). On MNIST, we decay the learning rate by a factor of 10 at the 250th and 350th epoch, and train for 400 epochs. On CIFAR-10, we train with cosine learning rate decay for a total of 200 epochs. All initial learning rates are 0.001. We report bits per dimension (bpd), which is proportional to negative log-likelihood.

Network Architecture For density estimation on MNIST, we use 20 grouped invertible layers with 45 filters each. For CIFAR-10, 16 grouped invertible layers are used, each of which has 255 filters. Two squeezing operations are used.

Method	MINIST	CIFAR10
MAF (Papamakarios et al., 2017)	1.89	4.31
Real NVP (Dinh et al., 2016)	1.06	3.49
Glow (Kingma & Dhariwal, 2018)	1.05	**3.35**
FFJORD (Grathwohl et al., 2018)	0.99	3.40
i-ResNet (Behrmann et al., 2019)	1.06	3.45
MinvNet (ours)	**0.98**	**3.35**

Table 1. MNIST and CIFAR10 bits per dimension results.

Figure 1. Samples from MinvNet. **Left:** MNIST samples, **Right:** CIFAR-10 samples.

Results We compare our results with previous arts in Tab. 1. MinvNet sets the new state-of-the-art bpd on MNIST and matches the previous state-of-the-art on CIFAR-10. Our MinvNet on MNIST has comparable number of parameters to i-ResNet, and our MinvNet on CIFAR-10 only uses $1/4$ parameters of i-ResNet. Note that all values in Tab. 1 are with respect to the continuous distribution of uniformly dequantized images, and results of models that view images as discrete distributions are not directly comparable (*e.g.*, PixelCNN (Oord et al., 2016), IAF-VAE (Kingma et al., 2016), and Flow++ (Ho et al., 2019)).

Sampling Samples can be obtained by sequentially inverting each layer of MinvNet. We use 5 iterations of the Newton-Raphson method to invert monotonic univariate functions. Uncurated samples are provided in Fig. 1.

6. Conclusion

We introduce MinvNet, an invertible network based on masked convolutions. We demonstrate its great potential in classification and density estimation. MinvNet's performance on classification is comparable to ResNet, indicating the strong expressive power of the architecture. For density estimation, it matches or surpasses the best reported results on MNIST and CIFAR-10.

References

Behrmann, J., Will Grathwohl, Ricky T. Q. Chen, D. D., and Jacobsen, J.-H. Invertible residual networks. *arXiv preprint arXiv:1811.00995*, 2019.

Berg, R. v. d., Hasenclever, L., Tomczak, J. M., and Welling, M. Sylvester normalizing flows for variational inference. *arXiv preprint arXiv:1803.05649*, 2018.

Dinh, L., Krueger, D., and Bengio, Y. NICE: non-linear independent components estimation. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Dosovitskiy, A. and Brox, T. Inverting visual representations with convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4829–4837, 2016.

Grathwohl, W., Ricky T. Q. Chen, Jesse Bettencourt, I. S., and Duvenaud, D. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016.

Ho, J., Chen, X., Srinivas, A., Duan, Y., and Abbeel, P. Flow++: Improving flow-based generative models with variational dequantization and architecture design, 2019. URL https://openreview.net/forum?id=Hyg74h05tX.

Jacobsen, J.-H., Smeulders, A. W., and Oyallon, E. i-revnet: Deep invertible networks. In *International Conference on Learning Representations*, 2018.

Jacobsen, J.-H., Behrmann, J., Zemel, R., and Bethge, M. Excessive invariance causes adversarial vulnerability. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=BkfbpsAcF7.

Kingma, D. P. and Dhariwal, P. Glow: Generative flow with invertible 1x1 convolutions. *arXiv preprint arXiv:1807.03039*, 2018.

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. Improved variational inference with inverse autoregressive flow. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 4743–4751. Curran Associates, Inc., 2016.

Loshchilov, I. and Hutter, F. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Mahendran, A. and Vedaldi, A. Visualizing deep convolutional neural networks using natural pre-images. *International Journal of Computer Vision*, 120(3):233–255, 2016.

Oord, A. V., Kalchbrenner, N., and Kavukcuoglu, K. Pixel recurrent neural networks. In Balcan, M. F. and Weinberger, K. Q. (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1747–1756, New York, New York, USA, 20–22 Jun 2016. PMLR. URL http://proceedings.mlr.press/v48/oord16.html.

Papamakarios, G., Pavlakou, T., and Murray, I. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pp. 2338–2347, 2017.

Phuong, T. T. and Phong, L. T. On the convergence proof of amsgrad and a new version. *arXiv preprint arXiv:1904.03590*, 2019.

Reddi, S. J., Kale, S., and Kumar, S. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=ryQu7f-RZ.

Rezende, D. and Mohamed, S. Variational inference with normalizing flows. In Bach, F. and Blei, D. (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1530–1538, Lille, France, 07–09 Jul 2015. PMLR. URL http://proceedings.mlr.press/v37/rezende15.html.

A. Proof of Theorem 1

Theorem 1. *If the following conditions hold for all* $\mathbf{x} \in \mathbb{R}^n$

- $J_f(\mathbf{x})$ *is lower triangular;*
- $\mathrm{diag}(J_f(\mathbf{x})J_f(0)) > 0$*,*

then f *is invertible. Moreover,* $\mathbf{x} = f^{-1}(\mathbf{y})$ *can be obtained by sequentially inverting* n *univariate monotonic functions.*

Proof. Let's first prove the following two lemmas.

Lemma 1. *Condition 1 implies* $f(\mathbf{x})_i$ *is a function of only* $\mathbf{x}_1, ..., \mathbf{x}_i$.

Proof. Due to the fact that $J_f(\mathbf{x})$ is lower triangular, we have $J_f(\mathbf{x})_{i,j} = \frac{\partial f(\mathbf{x})_i}{\partial \mathbf{x}_j} = 0$ for any $j > i$. This implies $f(\mathbf{x})_i$ is not a function of $\mathbf{x}_j$ for any $j > i$ and thus depends only on $\mathbf{x}_1, ..., \mathbf{x}_i$. □

Lemma 2. *For any* i*, condition 2 implies either of the following:*

i. $\frac{\partial f(\mathbf{x})_i}{\partial \mathbf{x}_i} > 0$ *for all* $\mathbf{x}_i \in \mathbb{R}$.

ii. $\frac{\partial f(\mathbf{x})_i}{\partial \mathbf{x}_i} < 0$ *for all* $\mathbf{x}_i \in \mathbb{R}$.

That is, $f(\mathbf{x})_i$ *is monotonic on* $\mathbf{x}_i$.

Proof. Clearly condition2 is equivalent to $\frac{\partial f(\mathbf{x}_i)}{\partial \mathbf{x}_i}\frac{\partial f(\mathbf{x}_i)}{\partial \mathbf{x}_i}|_{\mathbf{x}_i=0} > 0$ for any $\mathbf{x}_i$. This means for any $\mathbf{x}_i$, $\frac{\partial f(\mathbf{x}_i)}{\partial \mathbf{x}_i} \neq 0$ and shares the same sign as $\frac{\partial f(\mathbf{x}_i)}{\partial \mathbf{x}_i}|_{\mathbf{x}_i=0}$, which is a constant. This further implies $\frac{\partial f(\mathbf{x}_i)}{\partial \mathbf{x}_i}$ is either positive or negative for all $\mathbf{x}_i \in \mathbb{R}$, and $f(\mathbf{x})_i$ is thus monotonic on $\mathbf{x}_i$. □

We now incorporate the above lemmas to prove the theorem. To solve $\mathbf{x}_1$, we only need to solve $f(\mathbf{x})_1 = y_1$, which is a map defined only on variable $\mathbf{x}_1$. Since condition 2 implies $f(\mathbf{x})_1$ is monotonic on $\mathbf{x}_1$, we know $f(\mathbf{x})_1$ is injective, which implies that $f(\mathbf{x})_1$ has a unique inverse $\mathbf{x}_1$. Now assume we have already solved $\mathbf{x}_1, ..., \mathbf{x}_k$. To solve $\mathbf{x}_{k+1}$, let's plug in $\mathbf{x}_1, ..., \mathbf{x}_k$ to $f(\mathbf{x})_{k+1}$, which can then be treated as a real valued function defined only on $\mathbf{x}_{k+1}$. As condition 2 implies $f(\mathbf{x})_{k+1}$ is a monotonic function of $\mathbf{x}_{k+1}$, this tells us $\mathbf{x}_{k+1}$ is uniquely determined due to the injective property of monotonic function. We then repeat the process until $\mathbf{x}_n$. In this case, we show that $f^{-1}(y) = [\mathbf{x}_1, ..., \mathbf{x}_n]^\mathsf{T}$ exists, and can be solved uniquely. □