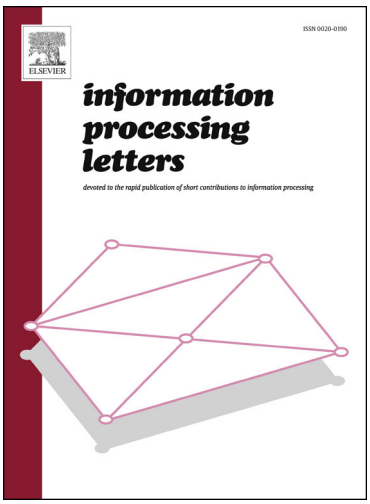
Accepted Manuscript



Lie group impression for deep learning

Mengduo Yang, Fanzhang Li, Li Zhang, Zhao Zhang

PII: S0020-0190(18)30061-9
DOI: https://doi.org/10.1016/j.ipl.2018.03.006
Reference: IPL 5660

To appear in: *Information Processing Letters*

Received date: 10 October 2016
Accepted date: 6 March 2018

Please cite this article in press as: M. Yang et al., Lie group impression for deep learning, *Inf. Process. Lett.* (2018), https://doi.org/10.1016/j.ipl.2018.03.006

Highlights

- Represent visual impression in a Lie group manifold way.
- Develop the single-layer Lie group model which is stacked to a deep neural network.
- Design a Lie group based gradient descent algorithm to train the network.

Lie group impression for deep learning

Mengduo Yang, Fanzhang Li, Li Zhang, Zhao Zhang

School of Computer Science and Technology, Soochow University, Suzhou, China

Corresponding author: Mengduo Yang, Email: mengduoyang@163.com

Abstract

In this work, we exploit a novel algorithm for capturing the Lie group manifold structure of the visual impression. By developing the single-layer Lie group model, we show how the representation learning algorithm can be stacked to yield a deep architecture. In addition, we design a Lie group based gradient descent algorithm to solve the learning problem of network weights. We show that our proposed technique yields representations that significantly better suited for training deep network and is also computationally efficient.

Keywords: visual impression; deep learning; Lie group

1. Impression and Deep Learning

Recently research of Yan & Huang who are the winners of PASCAL 2010 classification competition indicates that features are the key to the progress in recognition[1, 6]. And actually the description of features is an abstraction of the object needs to learn. Generally speaking, this kind of abstraction can be viewed as an impression[4]. For example, in a recognition task, one is given a picture and asked whether the animal in it is a cat. Usually, people can judge by their impressions of cats. For instance, in someone's impression, cats have whiskers and are furry with sharp claws. They can see in near darkness and can hear very faint sounds made by mice. In these impressions, some of them are easy to quantify and others are not. Since descriptions of features by computers are different from impressions of humans, computers cannot handle the abstract information if without a suitable model for those features hard to quantify[9]. Deep learning is a tool which can learn features automatically and uses a deep model with internal weights rather than a single value or vector to express those learned features[2, 8]. The structure of deep model can extract abstract features for subsequent classifiers[10].

Moreover, it is worth mentioning that descriptions of features are not same in

different levels or different views, which is similar to the idea of granular computing[11]. As a multi-level example, a blind man feels an elephant, only touching some part of it, and concluding what the elephant is like. The man draws an impression on the basis of partial understanding, and the overall judgement is composed of each part of the impression. We obtain various impressions based on one-side viewpoint and each view supplies amount of information which is merged into a concept by a certain structure. In addition, during the understanding of objective things, there is an idea of multilevel processing[5]. For example, when children learn the concept of "cat", they first notice various kinds of pictures and those pictures supply the pixel information from the lowest level, and then they will use this information to conclude several partial features of cats, such as arms and legs, head, whiskers and eyes. These features are established on the bottom pixels, and they are relatively more abstract and with a higher level. Up above, these features are continued to be combined to a much higher level of features, such as body shape and textures, until to the top decision level to determine whether the animal in the picture is a cat[3]. The idea of multilevel and multi-view is reflected in the deep learning algorithms. Thus, we can develop an abstract deep learning model with multilevel connection layers.

2. Lie group and Neural Network Learning

2.1 Single-Layer Lie Group Model

The training of a neural network is the training of weights which combine neurons of the network. Therefore, we develop the Lie group network weight training model.

Suppose we are going to train the weights associated with the connections between units in the layer i and units in the layer $i+1$. There are m units in the layer i and n units in the layer $i+1$. According to the functional mechanism of nonlinear neurons, the computation model of the hth unit in layer $i+1$ is

$$y_h = sigm\left(\sum_{j=1}^{m} w_{jh} x_j\right), h = 1,\ldots,n \qquad (1)$$

where $sigm(\cdot)$ is the activation function of neurons, the most common forms are $sigmoid$ and tanh function. w_{ij} is the weight associated with the connection between the jth unit in layer i and the hth unit in layer $i+1$. x_j is the output of unit j in the layer $i+1$. x_1 is the bias unit and the value is fixed at 1. Thus the general output of layer $i+1$ can be modeled as

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = sigm\left(\begin{pmatrix} w_{11} & \cdots & w_{1m} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & w_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} \right)$$

The above formula can be abbreviated as

$$Y = sigm(M^T X) \qquad (2)$$

where M is a $m \times n$ weight matrix, and $y_i \in (0,1), i = 1, \ldots, n$. According to the previous assumption, weight matrix can define a matrix set. This matrix set constitutes a sub-manifold in the Euclidean space $R^{m \times n}$ with dimension $m \times n$. This sub-manifold is called Stiefel manifold. Without any other constraint, this problem can be transformed into impose restrictions on the geometry structure of the matrix set as

$$M \in St(m,n) = \{W \in A_{m \times n} \mid W^T W = I_n\}$$

where $St(m,n)$ is a Stiefel manifold, and $A_{m \times n}$ represents all the $m \times n$ matrices, and I_n is the identity matrix with order n. Therefore, the learning problem of single layer weights in the network can be treated as the optimization problem

$$f(M) = E[u(X,Y,M)] \qquad (3)$$

where $f(M)$ represents the objective function of training, $E[\cdot]$ represents the expectation of the bracketed expression, and $u(\cdot,\cdot,\cdot)$ is a measurement of X, Y, M. Since the weight is matrix M, and during the training process, the updates of M are restricted on the Stiefel manifold. If we train the weight matrix M using gradient descent algorithm, the modification process of M is a geodesic along the Stiefel

manifold passing the point M and develop towards the steepest direction. Because it is difficult to solve the geodesic and gradient on the Stiefel manifold, we can transform M into the orthogonal matrix group $O(m)$. Suppose that $n \le m$, and this is reasonable because the dimension of network reduces gradually from input layer to output layer. Since $O(m)$ is a Lie group, we can use tools such as Lie algebra and exponential map to make an easy calculation. According to the result in [7], we can get the weight update formula

$$M_{i+1} = \tilde{M}_i \exp(-\eta \tilde{M}_i^T \tilde{V}_H) \begin{pmatrix} I_n \\ O_{(m-n,n)} \end{pmatrix}$$

$$\tilde{V}_H = gradf(M_i) = \frac{1}{2}(\nabla f(M_i) - M_i \nabla f(M_i) M_i) \qquad (4)$$

The above functions are the formulas to solve M_{i+1} according the gradient descent algorithm when we know the matrix M_i. Where $\tilde{M}_i = (w_1, \ldots, w_n, v_1, \ldots, v_{m-n}) \in O(m)$ represents $M_i = (w_1, \cdots, w_n) \in St(m,n)$, and $v_1, \cdots, v_{m-n}$ are used to let M_i form an orthogonal matrix. $O_{(m-n,n)}$ is a null matrix with $m-n$ rows and n columns. $\eta > 0$ is the step length, and $\nabla f(M_i)$ is the Euclidean gradient at M_i in optimization function f, i.e. $\nabla f(M_i) = \frac{\partial f(M_i)}{\partial (M_i)_{rc}}$. The denominator rc denotes take the partial derivatives of elements in each row and each column.

2.2 Learning Weights with Lie Group Model

With above foundations of mathematics, we can design a Lie group based gradient descent algorithm to solve the learning problem of network weights. The remaining problem is how to define the optimization function f and measurement u. If we use back propagation in multilayer deep learning neural networks, if may lead to a gradient diffusion problem. So we use an auto-encoder for layer-wise weight training. Given a group of input $\{X\}_{i=1}^{s}$, i.e. there are s examples. According to the single-layer Lie group model, we can obtain the encoder and decoder

Encoder: $Y = sigm(M^T X)$

Decoder: $\hat{X} = (M^T)^{-1} sigm^{-1}(Y)$ (5)

where $sigm^{-1}(\cdot)$ is the inverse function of $sigm(\cdot)$. Encoder encodes the vector X of the layer units into Y, while decoder executes the opposite operation, which returns output vector Y into the approximation $\hat{X}$ of data vector. A good choice of weight matrix M requires decoder restore the original data as much as possible. This can be obtained by measuring the difference between the original input and decoding result, i.e. the error of a single training example can be measured by the following formula

$$u(X,Y,M) = \left\| X - \hat{X} \right\| = \left\| X - (M^T)^{-1} sigm^{-1}(Y) \right\| \quad (6)$$

where $\|\cdot\|$ is the vector length, or the second moment norm in the Euclidean space. We plug formula (6) into (3), and obtain the total error function of the whole s examples, i.e. the objective function we need to optimize

$$\begin{aligned} f(M) &= E[u(X,Y,M)] \\ &= \frac{1}{s}\sum_{i=1}^{s} \left\| X_i - \hat{X}_i \right\| \\ &= \frac{1}{s}\sum_{i=1}^{s} \left\| X_i - (M^T)^{-1} sigm^{-1}(Y_i) \right\| \end{aligned}$$

Attention that M^T could not be a square matrix, so it is meaningless to use the signal $(M^T)^{-1}$. However, this signal is used to solve the inverse matrix of matrix M^T. If there is not any other condition, we can solve the pseudo-inverse of matrix M^T. In our problem, both M^T and M are points on the Stiefel manifold, and they meet the standard of $M^T M = I$. Therefore, we can use M to replace $(M^T)^{-1}$ in the previous formula, then we get

$$f(M) = \frac{1}{s}\sum_{i=1}^{s} \left\| X_i - M sigm^{-1}(Y_i) \right\| \quad (7)$$

According to the prototype function $sigm(x) = 1/(1+e^{-x})$, we can get the

inverse of the function $sigm^{-1}(x) = -\ln(1/x - 1)$. In order to solve the gradient of function f conveniently, we unfold the $f(M)$ then obtain

$$\begin{aligned} f(M) &= \frac{1}{s}\sum_{i=1}^{s}\left\| X_i - Msigm^{-1}(Y_i)\right\| \\ &= \frac{1}{s}\sum_{i=1}^{s}\left\| X_i - \begin{pmatrix} \sum_{j=1}^{n} w_{j1}\tilde{y}_{ij} \\ \vdots \\ \sum_{j=1}^{n} w_{jm}\tilde{y}_{ij} \end{pmatrix}\right\| \\ &= \frac{1}{s}\sum_{i=1}^{s}\left(\sum_{k=1}^{m}\left(x_{ik} - \sum_{j=1}^{n} w_{jk}\tilde{y}_{ij}\right)^2\right)^{\frac{1}{2}} \end{aligned} \tag{8}$$

where $\tilde{y}_{ij} = -\ln(\frac{1}{y_{ij}} - 1)$, and y_{ij} is the jth component in the output vector Y_i of the ith sample. According to the formula (8), we can obtain formula (9)

$$\nabla f(M) = \begin{pmatrix} \frac{\partial f(M)}{\partial w_{11}} & \cdots & \frac{\partial f(M)}{\partial w_{n1}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f(M)}{\partial w_{1m}} & \cdots & \frac{\partial f(M)}{\partial w_{nm}} \end{pmatrix} \tag{9}$$

To get every item in the above matrix, we can unfold the innermost bracket of the formula (8), i.e. the square term, then we obtain

$$\begin{aligned} f(M) &= \frac{1}{s}\sum_{i=1}^{s}\left(\sum_{k=1}^{m}\left(x_{ik}^2 + \left(\sum_{j=1}^{n} w_{jk}\tilde{y}_{ij}\right)^2 - 2x_{ik}\sum_{j=1}^{n} w_{jk}\tilde{y}_{ij}\right)\right)^{\frac{1}{2}} \\ &= \frac{1}{s}\sum_{i=1}^{s} v_i^{\frac{1}{2}}(X_i, Y_i, M) \end{aligned}$$

We use $v_i(X_i, Y_i, M)$, and we abbreviate $v_i(X_i, Y_i, M)$ to v_i in the later derivation.

$$\begin{aligned} v_i(X_i, Y_i, M) &= u_i^2 \\ &= \sum_{k=1}^{m}\left(x_{ik}^2 + \left(\sum_{j=1}^{n} w_{jk}\tilde{y}_{ij}\right)^2 - 2x_{ik}\sum_{j=1}^{n} w_{jk}\tilde{y}_{ij}\right) \\ &= \left\| X - \hat{X}_i\right\|^2 \end{aligned}$$

Then we get

$$\begin{aligned}\frac{\partial f(M)}{\partial w_{rc}} &= \frac{\partial f(M)}{\partial v}\cdot\frac{\partial v}{\partial w_{rc}} \\ &= \frac{1}{s}\sum_{i=1}^{s}\left(\frac{1}{2}v_i^{-\frac{1}{2}}\cdot\frac{\partial v_i}{\partial w_{rc}}\right)\end{aligned} \tag{10}$$

where $r=1,\cdots,n$ and $c=1,\cdots,m$, we also have

$$\frac{\partial v_i}{\partial w_{rc}} = 2\left(\sum_{j=1}^{n} w_{jc}\tilde{y}_{ij}\right)\cdot\tilde{y}_{ir} - 2x_{ic}\tilde{y}_{ir} \tag{11}$$

We plug formula (11) into (10), then we get

$$\begin{aligned}\frac{\partial f(M)}{\partial w_{rc}} &= \frac{1}{s}\sum_{i=1}^{s}\left(v_i^{-\frac{1}{2}}\left(\tilde{y}_{ir}\sum_{j=1}^{n} w_{jc}\tilde{y}_{ij} - \tilde{y}_{ir}x_{ic}\right)\right) \\ &= \frac{1}{s}\sum_{i=1}^{s}\left\|X_i-\hat{X}_i\right\|^{-1}\tilde{y}_{ir}\left(w_{\cdot c}\tilde{Y}_i - x_{ic}\right) \\ &= \frac{1}{s}\sum_{i=1}^{s}\frac{\tilde{y}_{ir}\left(\hat{x}_{ic}-x_{ic}\right)}{\left\|X_i-\hat{X}_i\right\|}\end{aligned} \tag{12}$$

where $v_i^{-\frac{1}{2}} = \left\|X_i-\hat{X}_i\right\|^{-1}$, $\tilde{Y}_i=(\tilde{y}_{i1},\cdots,\tilde{y}_{in})^T$, $w_{\cdot c}$ is the cth row in the matrix M or the cth column of the matrix M^T, $\hat{x}_{ic}$ is the cth component of the ith approximation vector $\hat{X}_i$. Thus in the iterative process of the algorithm, we can easily to solve $\nabla f(M_i)$ according to formula (12). Then we can use formula (4) to solve M_{i+1} iteratively, until the optimization function $f(M)$ is smaller than the given value or get the maximum iterative number.

3. Algorithms and analysis

Algorithm1. Single-layer Lie group learning algorithm

Input: $\{X\}_{i=1}^{s}$, X_i denotes the ith sample, and there are s input samples. The number of neurons in the output layer is identical with the number of neurons in the input layer.

Output: Weight matrix M of neural network.

Step1. Create a neural network with m input units, n hidden units and m output units.

Step2. Initialize the network weights, and make $M^T M = I$.

Step3. Before meet the end condition:

a) Input sample X_i into network, and compute the output units of network using $Y_i = sigm(M^T X_i)$.

b) Compute the error function of network $f(M)=\frac{1}{s}\sum_{i=1}^{s}\left(\sum_{k=1}^{m}\left(x_{ik}-\sum_{j=1}^{n} w_{jk}\tilde{y}_{ij}\right)^2\right)^{\frac{1}{2}}$.

c) Use formula (12), compute gradient $\nabla f(M_i)=\frac{1}{s}\sum_{i=1}^{s}\frac{\tilde{y}_{ir}\left(\hat{x}_{ic}-x_{ic}\right)}{\left\|X_i-\hat{X}_i\right\|}$ of every weight in the network.

d) Update the whole network weights $M_{i+1}=\tilde{M}_i \exp(-\eta \tilde{M}_i^T \tilde{V}_H)\begin{pmatrix} I_n \\ O_{(m-n,n)} \end{pmatrix}$ by formula (4), where $\tilde{V}_H = gradf(M_i)=\frac{1}{2}(\nabla f(M_i)-M_i \nabla f(M_i) M_i)$.

When each sample is trained in the single-layer neural network with m input nodes and n hidden nodes, the time complexity of Algorithm1 is $O(mn)$. This is because that for each sample, the network needs to construct the connections of both encoder and decoder during initialization, the time cost is $mn+nm$. In step3, the calculated amount of activation value is mn, and the calculated amount of error function is mns. Besides, the calculated amounts of gradient and update of the network are mns and mn. Suppose that the iteration number is k, for situation of single example $s=1$, the calculated amount of the whole algorithm is $mn+mn+k(mn+mns+mns+mn)$, and the time complexity is $O(mn)$.

Algorithm2. Lie group impression deep learning algorithm

Input: s training example $\{X\}_{i=1}^{s}$. The network structure with m input units, l hidden layers, n output units, and the number of hidden units in each hidden layer $\{n_j\}_{j=1}^{l}$.

Output: Weight matrices $\{M_j\}_{j=1}^{l+1}$ of the network, and the output layer.

Step1. Before meet the output layer

a) If $n_j = 1$, input s training samples $\{X\}_{i=1}^{s}$, and use algorithm1 to get M_1.

b) If $1 < n_j \le l$, compute $sigm(\tilde{M}_{n_l-1}{}^T X_i), i = 1,\ldots,s$ as input, and use algorithm1 to get $\{M_j\}_{j=2}^{l+1}$.

Step2. Fine tune the whole neural network in a supervised way.

Step3. Output the weight matrices $\{M_j\}_{j=1}^{l+1}$, and the output layer.

For a deep neural network with m input nodes, l hidden layers and n hidden nodes, the time complexity of algorithm2 is $O(mn + n^2)$. This is because that when we construct the neural network with algorithm2, the calculated amount of initialization is $mn_1 + n_1n_2 + \cdots + n_{l-1}n_l$. The calculated time of sparsification and orthogonalization is $s = O(mn_1 + n_1n_2 + \cdots + n_{l-1}n_l)$ and $o = O(mn_1 + n_1n_2 + \cdots + n_{l-1}n_l)$. When a single sample is trained on the neural network, the time of updating weights of each layer is $t = mn_1 + mn_1 + n_1n_2 + n_1n_2 + \cdots\cdots + n_{l-1}n_l + n_{l-1}n_l$. Suppose the iteration number is k, the time is $O(k(mn_1 + n_1n_2 + \cdots + n_{l-1}n_l + s + o + t + r)) = O(mn_1 + n_1n_2 + \cdots + n_{l-1}n_l)$. Because of the sparsification, the actual acting nodes $n_i, i = 1,\ldots,l$ of hidden layers are determined by the number of activation nodes. To simplify, we suppose the number of hidden nodes is $n_i = n, i = 1,\ldots,l$, then the time complexity of Lie group impression deep learning algorithm is $O(mn + (l-1)n^2) = O(mn + n^2)$.

4. Experiments

In our experiments, we test the ability of neural network to generalize from small scale training samples with the help of Lie group impression. We still limit the weight parameters of network to a Lie group structure. Especially when restrict the weight parameters, Lie group impression deep learning algorithm can supply a much smaller searching space than the traditional deep learning algorithm. In MNIST dataset, we compare the classification performance and the convergence rate of them.

MNIST is a dataset of 28×28 images of handwritten digits. The learning task is to predict the digit contained in the images. The MNIST dataset consists of 60000

training examples and 10000 testing examples. For both Lie group impression model and stacked auto-encoder deep learning model, we use a softmax classifier to compare the recognition results.

Figure1 shows the results of stacked auto-encoder in both cases, one is with the fine tuning and the other is without the fine tuning. The iterative numbers are 10, 15, 20, 25, 30, 35, 40, 45, 50, 100, 200, 300, and 400. From the results, we can see that fine tuning makes a huge impact in promoting the recognition rates by about 10%.

Figure2 shows the results by Lie group impression model in the same iterative condition. Compared to the stacked auto-encoder, the training results are close to the results with fine tuning. This can reflect that the manifold constraint makes the searching initial value close to some local optimum during the network training. From figure1 we can also see local fluctuation when the iterative number is lack. This reflects that the network parameters are unstable during the searching process and less iterative number cannot help the network to stabilize a high classification performance.

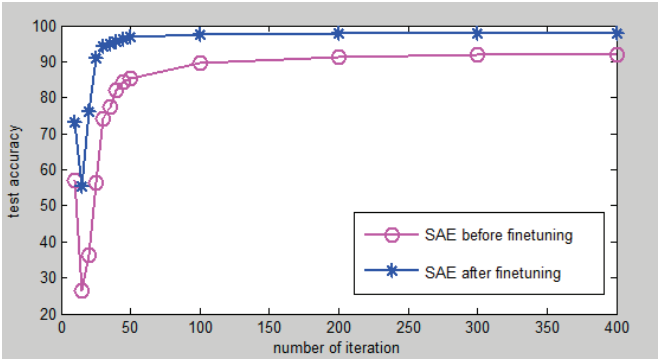


Figure1

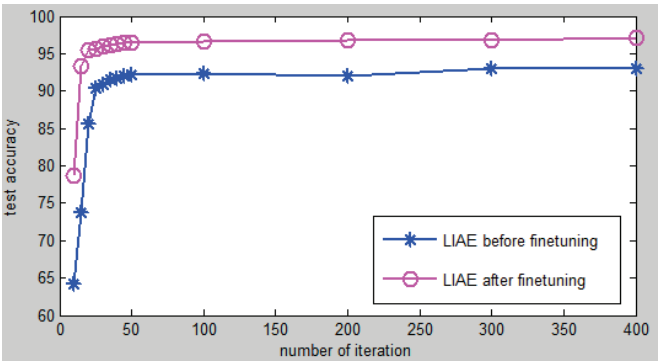


Figure2

Another experiment compares the performance of Lie group impression model and stacked auto-encoder model with the help of fine tuning. From figure3 we can see that Lie group impression model has a faster iterative rate and is able to get the optimal classification performance with a much less iterative number.

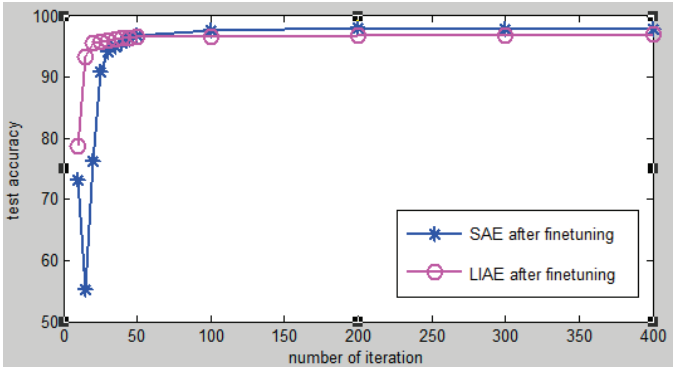


Figure3

5. Conclusions

It is easier and faster for computers to recognize target objects by visual impression. This paper learns visual impression with Lie group structure during the training process of neural network by introducing the concept of Stiefel manifold. The constraint of network parameters greatly reduces the value range of parameters space, which is an outstanding advantage compared to the traditional deep learning algorithms. Experiment results further proved that Lie group impression deep learning model is a feasible method. It supplies a new approach to extract features for image recognition by deep learning methods. Based on the research results of this paper, the geometry structure of parameter space needs more in-depth research, which may help to bring a better classification performance.

Acknowledgements

This work was supported by National Nature Science Foundation of China [61672364, 61672365 and 61033013].

References

1. Bengio, Y., A. Courville, and P. Vincent, *Representation Learning: A Review and New Perspectives.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 2013. **35**(8): p. 1798-1828.

2. Bengio, Y., *Learning deep architectures for AI.* Foundations and trends in Machine Learning, 2009. **2**(1): p. 1-127.
3. DiCarlo, J.J., D. Zoccolan, and N.C. Rust, *How does the brain solve visual object recognition.* Neuron, 2012. **73**(3): p. 415-434.
4. Higgins, I., L. Matthey, and X. Glorot, *Visual Concept Learning with Unsupervised Deep Learning.* arXiv preprint, 2016. **1606.05579**.
5. Hinton, G.E., *Learning multiple layers of representation.* Trends in cognitive sciences, 2007. **11**(10): p. 428-434.
6. Krizhevsky, A., l. Sutskever, and G.E. Hinton, *Imagenet classification with deep convolutional neural networks*, in *Advances in neural information processing systems*2012. p. 1097-1105.
7. Nishimori, Y., *A Neural Stiefel Learning based on Geodesics Revisited.*
8. Szegedy, C., *An Overview of Deep Learning.* AITP, 2016. **2016**.
9. Yokota, S., D. Chugo, and H. Hashimoto, *Visual impression to robot motion imitating human-study on delay motion*, in *2016 9th International Conference on Human System Interations*2016. p. 435-439.
10. Zeiler, M.D., G.W. Taylor, and R. Fergus, *Adaptive deconvolutional networks for mid and high level feature learning*, in *IEEE International Conference on Computer Vision*2011. p. 2013-2025.
11. Zeiler, M.D., et al., *Deconvolutional Networks*, in *Computer Vision and Pattern Recognition*2010. p. 2528-2535.