Motion Sequence Recognition with Multi-sensors Using Deep Convolutional Neural Network

Runfeng Zhang and Chunping Li

Abstract With the rapid development of intelligent devices, motion recognition methods are broadly used in many different occasions. Most of them are based on several traditional machine learning models or their variants, such as Dynamic Time Warping, Hidden Markov Model or Support Vector Machine. Some of them could achieve a relatively high classification accuracy but with a time-consuming training process. Some other models are just the opposite. In this paper, we propose a novel designed deep Convolutional Neural Network (DBCNN) model using "Data-Bands" as input to solve the motion sequence recognition task with a higher accuracy in less training time. Contrast experiments were conducted between DBCNN and several baseline methods and the results demonstrated that our model could outperform these state-of-art models.

1 Introduction

Nowadays, intelligent devices and modern life are more and more inseparable. These devices could record humans' motion sequences through embedded sensors precisely. Lots of motion recognition models have been proposed, such as Hidden Markov Models, Dynamic Time Warping, and Feature-Based Support Vector Machine. There are drawbacks more or less. Some of them can not attain a high accuracy, others might be time-consuming. In this paper, we proposed a new model that pre-processes motion sequences to form "data-bands" and then trains special designed deep convolutional neural network using "data-bands" as inputs. We call this model as DBCNN, in which "DB" means "Data-Bands" and "CNN" means "Convolutional Neural Network". Experiments show that our novel model could attain a better recognition accuracy compared with other state-of-art methods. Due to

R. Zhang (✉) · C. Li
School of Software, Tsinghua University, Beijing 100084, BJ, China
e-mail: rf-zhang12@mails.tsinghua.edu.cn

C. Li
e-mail: cli@tsinghua.edu.cn

A. Abraham et al. (eds.), *Intelligent Data Analysis and Applications*,
Advances in Intelligent Systems and Computing 370,
DOI 10.1007/978-3-319-21206-7_2

characteristics of deep convolutional neural network, training process can be accelerated on GPU in extremely short time, which makes DBCNN win more scores.

This paper is organized as follows. Section 2 introduces some backgrounds, including problem description, related works and some basic knowledge about Convolutional Neural Network (CNN) in brief. Section 3 presents DBCNN for motion recognition task, including data process and deep Convolutional Neural Network design. Section 4 takes a glance at some details of parameters learning. Section 5 reports experiment results of our model comparing with two other state-of-art methods. And Sect. 6 concludes this paper.

2 Background

2.1 Problem Description

Now we describe motion recognition task more formally : Motion sequence recognition is a classification task with $\mathcal{N}$ categories, and each of these categories has a corresponding training set $\mathcal{D}_i$. The total training set is $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_{\mathcal{N}}\}$. Training set $\mathcal{D}_i$ has $\mathcal{N}_i$ action instances: $\mathcal{D}_i = \{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_{\mathcal{N}_i}\}$. As mentioned before, each motion instance is constituted by several sequences gathered by different sensors, so we have $\mathcal{M}_i = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{channel\_num}\}$. Assuming that processed sequences have t dimension, sequence $\mathcal{S}_i$ would be a vector with length t. The motion recognition task is to build a classifier given training dataset $\mathcal{D}$.

2.2 Related Works

A wide variety of motion sequence recognition models have been proposed, such as Hidden Markov Models (HMMs) [1–4], Dynamic Time Warping (DTW) [5], and Feature-Based Support Vector Machine (Feature-SVM) [6] etc. Among models of HMMs, continuous-HMM proposed in [3] claimed to gain the highest accuracy. In a task with 10 pre-setting motions, it got a classification accuracy of 96.76 %. This result has beaten the discrete-HMM [4], which could achieve an accuracy of 96.1000 % in the same dataset. Liu et al. has given a better result than both discrete-HMM and continuous-HMM in [5] using Dynamic Time Warping. SVM could not be used in our task directly due to the large scale and high dimensionality of sequences. Wu et al. [6] proposed a feature-based SVM algorithm, which extracts new features with a lower dimension from the original sequences in a manual procedure. Wang et al. [7] proposed a similar method in different feature extraction rules. Experiment results in [7] show that Wang's method could achieve higher accuracy comparing with HMMs [1–4], DTW [5] and Feature-SVM designed by Wu et al. [6], we adopt this Feature-SVM model as one of our baseline methods in our experiment.

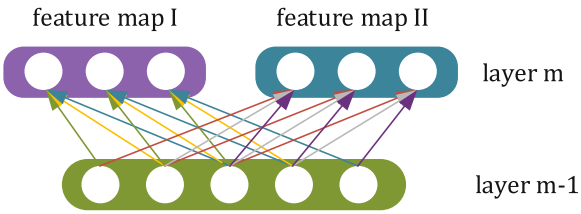

Fig. 1 Shared weights of convolutional layers in CNN

2.3 Convolutional Neural Network

Convolutional Layer in CNN Although full connection structure in multi-layer perception could make the networks to have more ability in expression, model will be more possible to overfit the data. CNN solves these problems via sparse connection structure, as shown in Fig. 1. First, nodes in high layer only connect to parts of nodes in low layer, which could reduce the number of parameters efficiently. Moreover, nodes in high layer share identical weights. Just like Fig. 1, arrows with the same weight value have identical colors. This structure make the model to detect position independent features, from which we benefit a lot.

Sub-sampling Layer in CNN Pooling is a form of non-linear down-sampling in CNN. Among several different pooling methods, max-pooling and average-pooling are the most common. LeCun et al. [9] used this method in their LeNet-5 for two reasons: reducing the computational complexity for upper layers and providing a form of translation invariance.

3 The DBCNN Model for Motion Sequence Recognition

3.1 Data Preprocess

Different people might do the same motion in different duration time. Sequences of motion gathered by intelligent devices are usually sampled in a constant time interval. So our obtained sequences of motion instances have different dimensionality.

To overcome this, our model introduces an approximation, linear interpolation. We get values of a sequence in specific proportional position to form new feature vectors, which have a fixed dimensionality. Actually, the value in specific proportional position is a linear combination of two real sample values near the position.

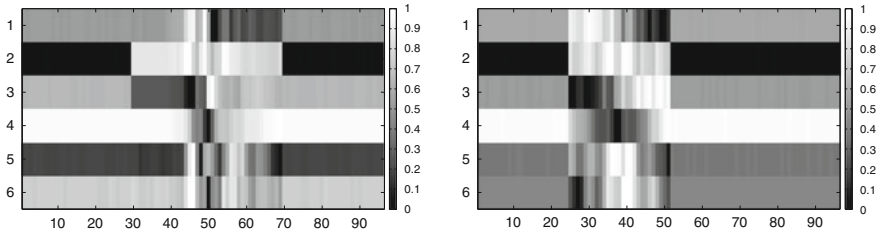

Fig. 2 Data band of the motion swipe right

3.2 Data-Band

Now, the new sequences of motion instances are in a same dimension. In our experiment, each instance of motion data consists of six sequences. Instead of arranging the six sequences into a long vector, we organize them into a "data-band", which is a narrow matrix with a small row number and a large column number, just like a band.

Figure 2 shows "data-bands" of two motion instances belong to a same motion category, "Swipe Right". It is obvious that they have position invariance: there are similar parts appearing in distinct positions. This invariance in a same motion class is an inherent attribute of motion itself and irrelevant to the motion's duration or executors. Our model use "data-bands" as input could grasp the position invariant successfully, which could not be achieved if we organize the data into a long vector.

3.3 Specially Designed CNN for Recognition of Motion Sequence

Input of CNN In our proposed CNN for sequences of motion, the input of network are actually "data-bands" as mentioned before. In Fig. 3, the input of the network is showed in the left side clearly. These "data-bands" are matrices with dimensionality of 6×96. The rows of matrices correspond to 6 axes (e.g. x, y, z axes in both accelerator and angular speed), and the columns correspond to 96 sampled points.

Design of Convolutional Layer LeCun et al. [9] have shown that CNN model is extremely suitable for data with high dimensionality and invariant attributes. Sequences of motion instance just fit the properties mentioned above. However, in LeNet-5, receptive fields are little squares to capture the invariance between images pixels. In our model, we treat our pre-processed sequences, "data-bands", as images. The difference is that the invariance of "data-bands" exhibits in a narrow form spreading in the lengthways dimension in "data-bands" matrices. If we use small squares as receptive fields, more receptive fields are necessary to catch the invariance. At the same time, the CNN model has more parameters and is more possible to be overfitting. Our model solves the invariance extraction problem in a very elegant

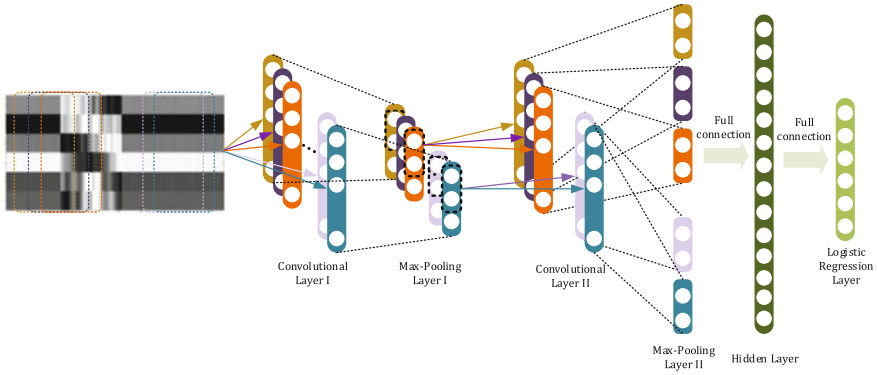

Fig. 3 The DBCNN model for sequences of motion data

way: we set up several convolutional windows to cover the lengthways of "data-bands" matrices, just like dashed rectangles with distinct colors in left of Fig. 3. This design could catch the invariance in lengthways using relatively small number of receptive fields. Moreover, each convolutional window scans a "data-band" and could get a 1-D feature vector, not a matrix in general CNN for image recognition.

Pooling Layer A pooling layer provides a further dimensionality reduction. We concatenate a max-pooling layer after a convolutional layer to shrink the feature space. At the same time, pooling layer provides an auxiliary way to grasp position invariant feature.

Classifier After two convolution-pooling structures, we connect a hidden layer and a logistic regression layer. They form a general multi-layer perception. In fact, two convolution-pooling layers mentioned before could be regarded as a function of feature transformation.

4 Details of Learning

4.1 Gradients in Convolutional Layers

In convolutional layers, a small patch $(\boldsymbol{p}_i^{l-1})_{uv}$ in previous layer's feature maps $\boldsymbol{x}_i^{l-1}$ are convolved with kernels $\boldsymbol{k}_{ij}^l$ and added a bias b_i^l to get a value $(\boldsymbol{u}_j^l)_{uv}$, then activation function $f(\cdot)$ would act on this value to form the corresponding element at (u, v) in the output feature maps $\boldsymbol{x}_i^l$, which is noted by $(\boldsymbol{x}_j^l)_{uv}$:

$$(\boldsymbol{u}_j^l)_{uv} = \sum_{i \in M_j} \boldsymbol{p}_i^{l-1} * \boldsymbol{k}_{ij}^l + b_j^l, \tag{1}$$

$$(\boldsymbol{x}_j^l)_{uv} = f((\boldsymbol{u}_j^l)_{uv}). \tag{2}$$

In Eq. 1, M_j means the selection of input maps in layer $l-1$. It is decided by the network structure when you design it. $\boldsymbol{p}_i^{l-1}$ and $\boldsymbol{k}_{ij}^l$ are two matrices, and the "$*$" operation means to do element-wise multiplication and get the sum of the new matrix. Generally, there is an objective error function E for a neural network to optimize the parameters. In order to compute derivatives of parameters on the error function E, we could firstly compute sensitivity $\boldsymbol{\delta}_j^l$ defined as follows:

$$(\boldsymbol{\delta}_j^l)_{uv} = \frac{\partial E}{\partial (\boldsymbol{u}_j^l)_{uv}}. \tag{3}$$

Sensitivity computation is a top-down process similar to the situation in general neural network but with a little difference. We compute sensitivity $(\boldsymbol{\delta}_j^l)_{uv}$ in layer l via the sensitivity $(\boldsymbol{\delta}_j^{l+1})_{uv}$ in layer $l+1$:

$$\boldsymbol{\delta}_j^l = \beta_j^{l+1}(f'(\boldsymbol{u}_j^l) \circ up(\boldsymbol{\delta}_j^{l+1})), \tag{4}$$

where $up(\cdot)$ function is exactly an inverse process of the down-sampling and β_j^{l+1} is a parameter in next sub-sampling layer (please refer to sect. 4.2). In Eq. 4, we use the notation "$\circ$" to indicate the element-wise multiplication of two matrices. After getting sensitivities for feature maps in convolutional layer, we could compute gradients about parameters in convolutional layers as follows:

$$\frac{\partial E}{\partial \boldsymbol{k}_{ij}^l} = \sum_{u,v} (\boldsymbol{\delta}_j^l)(\boldsymbol{p}_i^{l-1})_{uv}, \tag{5}$$

$$\frac{\partial E}{\partial b_j^l} = \sum_{u,v} (\boldsymbol{\delta}_j^l)_{uv}. \tag{6}$$

4.2 Gradients in Sub-sampling Layers

Sub-sampling layers don't change the number of input maps, and they are really down-sampled versions which would downsize the output maps as follows:

$$\boldsymbol{x}_j^l = f(\beta_j^l down(\boldsymbol{x}_j^{l-1}) + b_j^l). \tag{7}$$

Here, $down(\cdot)$ means the sub-sampling function, such as average-pooling or max-pooling. The parameter β_j^l is a scalar to control the efficiency of sub-sampling, and b_j^l is still a bias item just like in convolutional layers. To compute the gradients about

β_j^l and b_j^l, we adopt similar steps to compute the sensitivities and gradients:

$$\boldsymbol{\delta}_j^l = f'(\boldsymbol{u}_j^l) \circ \boldsymbol{q}_j^l. \tag{8}$$

In Eq. 8, $\boldsymbol{q}_j^l$ is a matrix which has the same size as $\boldsymbol{\delta}_j^l$, and its element at (u, v) is noted by $(\boldsymbol{q}_j^l)_{uv}$, which has a complex definition form as follows:

$$(\boldsymbol{q}_j^l)_{uv} = \sum_{i \in \Phi(j)} \sum_{s,t} (\boldsymbol{k}_{i,j}^{l+1})_{s,t} (\boldsymbol{\delta}_i^{l+1})_{u'(s,t),v'(s,t)}, \tag{9}$$

where $\Phi(j)$ is a layer $l+1$ feature maps' set for which feature map j in layer l, $\boldsymbol{x}_j^l$, is participated the convolutional process. $(\boldsymbol{\delta}_i^{l+1})_{u'(s,t),v'(s,t)}$ is a tricky representation that means an element in feature map $\boldsymbol{x}_i^{l+1}$ which use the element $\boldsymbol{x}_j^l)_{uv}$ during its convolutional process with weight $(\boldsymbol{k}_{i,j}^{l+1})_{s,t}$. It is worth to notice that if the element corresponding to weight $(\boldsymbol{k}_{i,j}^{l+1})_{s,t}$ does not exist, the value of $(\boldsymbol{\delta}_i^{l+1})_{u'(s,t),v'(s,t)}$ would be zero.

Now, we could compute the gradients for b_i^l and β_j^l as the following equations:

$$\frac{\partial E}{\partial b_j^l} = \sum_{u,v} (\boldsymbol{\delta}_j^l)_{uv}, \tag{10}$$

$$\boldsymbol{d}_j^l = down(\boldsymbol{x}_j^{l-1}), \tag{11}$$

$$\frac{\partial E}{\partial \beta_j^l} = \sum_{u,v} (\boldsymbol{\delta}_j^l \circ \boldsymbol{d}_j^l)_{uv}. \tag{12}$$

After combining all the details mentioned before, we have our parameters optimization algorithm for CNN in **Algorithm 1**.

5 Experiment Results

5.1 Introduction to 6DMG

Our experiments are based on the dataset 6DMG [8], which is a 6-D Motion gesture database developed by Chen et al. in Georgia Institute of Technology. In our experiments, we use parts of original sequences: acceleration sequences (including $x, y, z-$axies) and angular speed sequences (including $x, y, z-$axies). Figure 4 shows 20 kinds of motion in 6DMG. The size of training set is 32300. The size of validation set is 4000 and the one of test set is 5100.

Algorithm 1 Parameters Optimization for Convolutional Neural Network

Require: processed dataset D; learning rate α; epoch number $\mathcal{N}$; batch size n_b; Network Structure $\mathcal{M}$
Ensure: CNN's parameters $\boldsymbol{W, k, b, \beta}$
initialize network parameters $\boldsymbol{W, k, b, \beta}$ randomly
for each $i \in [1, \mathcal{N}]$ **do**
for each batch data $\mathcal{D}_i$ with size n_b **do**
for layer $l := L$ to 1 **do**
1. compute **sensitivities** of layer l according to back-propagation
2. compute **gradient** of each parameters in layer l
3. **update parameters** in layer l in gradient descent method with learning rate α
end for
end for
end for

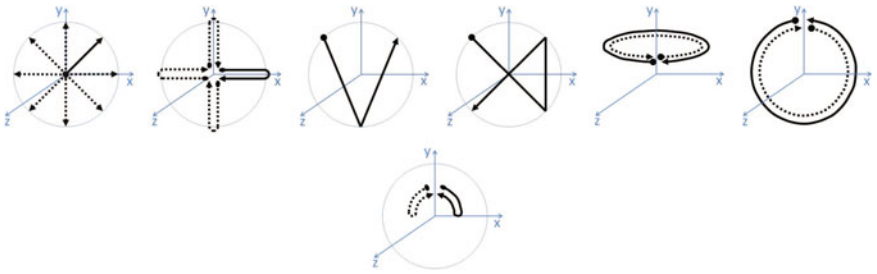

Fig. 4 Twenty kinds of motion in 6DMG

5.2 Feature-SVM

Feature-SVM proposed by Wang et al. [7] is claimed to outperform HMMs [1–4], DTW [5] and another featured-SVM given by Wu et al. [6]. We adopt this model as one of baseline experiments. After feature transformation, new features have the length of 48. We list the feature design rules in Table 1.

New 48-d features are trained using Gaussian Kernel SVM and Linear Kernel SVM with a 5-fold cross validation. For Linear Kernel SVM, the best accuracy 96.9125 % is attained when the cost parameter $log_2 C$ is 2.3206. And for Gaussian Kernel SVM, the best result appears when $log_2 C$ is 1.9459 and $log_2 g$ is -2.8332, and the best classification accuracy is 93.7011 %.

5.3 Deep Belief Network

Hinton et al. [14] proposed a fast learning algorithm for deep belief network, which use a greedy layer-wise training method. Deep architecture could be built with blocks (e.g. restricted boltzmann machine). Parameters in DBN would be fine-tuned finally.

Table 1 Feature extraction map in feature-SVM

Stochastic characteristic	Dimension
Acceleration (x, y, z) and angular speed (x, y, z) mean	6
Acceleration and angular speed standard deviation	6
Acceleration and angular speed variance	6
Acceleration and angular speed interquartile range	6
Acceleration correlation coefficient	3
Angular speed correlation coefficient	3
Acceleration and angular speed MAD	6
Acceleration and angular speed root mean square	6
Acceleration and angular speed energy (mean square of DFT items)	6
Total	48

Table 2 Deep belief network's results for motion sequence recognition

Learning rate	DBN structure	1000 epoch accuracy (%)	2000 epoch accuracy (%)
0.10	[600, 600]	97.5736	97.5736
	[1000, 1000]	97.5713	97.5713
	[600, 600, 600]	97.7295	97.7736
	[600, 600, 600, 600]	97.7104	97.7210
0.03	[600, 600]	97.3512	97.6059
	[1000, 1000]	97.4776	97.7585
	[600, 600, 600]	97.8736	**97.9571**
	[600, 600, 600, 600]	97.7753	97.8560
0.01	[600, 600]	97.3340	97.5913
	[1000, 1000]	97.3933	97.5407
	[600, 600, 600]	97.5852	97.7107
	[600, 600, 600, 600]	97.5115	97.6781

We get the best result with accuracy of 97.9571 % using the DBN model. The structure with the highest classification accuracy has 3 hidden layers, and each layer has 600 nodes. Learning rate in the process is 0.03 with 20 % attenuation every 500 epochs. Table 2 records experiment results in more details.

5.4 The DBCNN Model

Table 3 gives the classification accuracy under distinct parameters configuration. As we could see, our DBCNN model has achieved a recognition accuracy of 98.66 %, which outperforms the other two methods. Left part of Fig. 5 gives a results comparison among these three models.

Table 3 DBCNN's results for motion sequence recognition

#Kernels	1st conv filter	1st pooling	2nd conv filter	2nd pooling	Accuracy (%)
[20,40]	25×6	3×1	5×1	2×1	98.06
[30, 40]	25×6	3×1	5×1	2×1	98.22
[30, 50]	25×6	3×1	5×1	2×1	98.40
[40, 40]	25×6	3×1	5×1	2×1	**98.66**
	33×6	4×1	5×1	2×1	98.57
[40, 50]	25×6	3×1	5×1	2×1	98.58
[45, 35]	25×6	3×1	5×1	2×1	98.42
[45, 45]	25×6	3×1	5×1	2×1	98.52

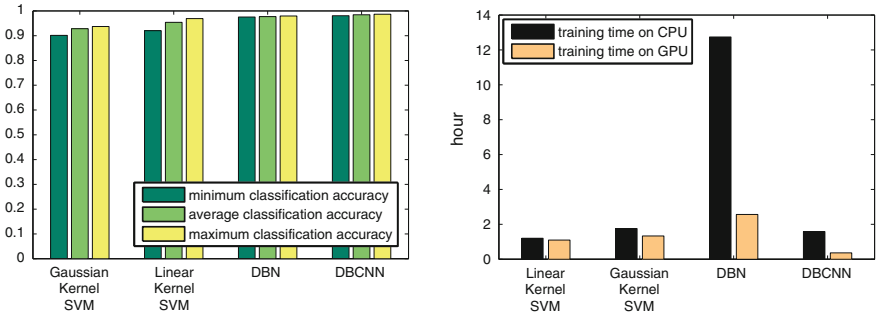

Fig. 5 Recognition accuracy comparison and training time comparison

In our experiments, a GTX-660 graphic unit with 2G GPU RAM is used. Training time using DBCNN model is about 22 min on GPU. As a contrast, it takes about 3 times longer to train the Feature-SVM model on GPU, and about 7 times longer to train the DBN model on GPU. Training Feature-SVM model and DBN model on CPU might take much longer time. Right part of Fig. 5 shows an approximate training time of these models on CPU and GPU.

6 Conclusion

In this paper, we proposed a novel motion sequence recognition model, DBCNN. First of all, we pre-process motion sequences to form "data-bands". Then we train a specially designed convolutional neural network using these "data-bands" as input. Our experiments on 6DMG demonstrate that our model can get a higher recognition accuracy than state-of-art algorithms in less training time.

References

1. Amma C, Gehrig D, Schultz T (2010) Airwriting recognition using wearable motion sensors. In: Proceedings of the 1st augmented human international conference. ACM
2. Schlömer T et al (2008) Gesture recognition with a Wii controller. In: Proceedings of the 2nd international conference on Tangible and embedded interaction. ACM
3. Timo P (2005) Accelerometer based gesture recognition using continuous HMMs. Springer, Berlin, pp 639–646 (Pattern Recognit Image Anal)
4. Kallio S, Juha K, Jani M (2003) Online gesture recognition system for mobile interaction. In: IEEE International conference on systems, man and cybernetics
5. Liu J et al (2009) uWave: accelerometer-based personalized gesture recognition and its applications. Pervasive Mobile Comput 5:657–675
6. Wu J et al (2009) Gesture recognition with a 3-d accelerometer. Ubiquitous intelligence and computing. Springer, Berlin, pp 25–38
7. Wang J-S, Chuang F-C (2012) An accelerometer-based digital pen with a trajectory recognition algorithm for handwritten digit and gesture recognition. IEEE Trans Ind Electron 2998–3007
8. Chen M, AlRegib G, Juang B-H (2012) A new 6d motion gesture database and the benchmark results of feature-based statistical recognition. In: IEEE international conference on emerging signal processing applications
9. LeCun Y et al (1998) Gradient-based learning applied to document recognition. Proc IEEE 2278–2324
10. Cho S-J et al (2006) Two-stage recognition of raw acceleration signals for 3-D gesture-understanding cell phones. In: Tenth international workshop on frontiers in handwriting recognition
11. Bishop CM (1995) Neural networks for pattern recognition
12. Bengio Y (2009) Learning deep architectures for AI. In: Machine learning foundations trends, pp 1–127
13. Hinton GE (2007) Learning multiple layers of representation. Trends Cognit Sci 428–434
14. Hinton G, Osindero S, Teh Y-W (2006) A fast learning algorithm for deep belief nets. Neural Comput 1527–1554
15. Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. Science 504–507
16. Hinton GE (2012) A practical guide to training restricted boltzmann machines. In: Montavon G, Orr GB, Müller K-R (eds) Neural networks: tricks of the trade, vol 7700, 2nd edn. LNCS. Springer, Heidelberg, pp 599–619
17. Bergstra J et al (2011) Theano: Deep learning on gpus with python. In: BigLearning workshop, NIPS
18. Bengio Y et al (2007) Greedy layer-wise training of deep networks. In: Advances in neural information processing systems
19. Chang C-C, Lin C-J (2011) LIBSVM: a library for support vector machines. ACM Trans Intell Syst Technol

http://www.springer.com/978-3-319-21205-0

Intelligent Data Analysis and Applications
Proceedings of the Second Euro-China Conference on Intelligent Data Analysis and Applications, ECC 2015
Abraham, A.; Jiang, X.H.; Snasel, V.; Pan, J.-S. (Eds.)
2015, XX, 560 p. 196 illus., Softcover
ISBN: 978-3-319-21205-0