Deep Learning in robotics : 3D object recognition and modeling human motion

Hugues Porneuf, Maëlys Robert, Jinhai Zhou

Télécom Bretagne

name.surname@telecom-bretagne.eu

Abstract— **Deep Learning is a research trend in a lot of famous laboratories. Deep Learning is often associated with Artificial Intelligence because its performance on computer vision, speech recognition and sentiment analysis is very powerful. Moreover, Artificial Intelligence is linked with robotics, especially the development of domestic robots. Hence a great amount of application of Deep Learning algorithms is applied to allow robots to evolve in an every-day life environment and to interact effectively with people. Deep learning technology is being used in many aspects of modern society. Compared with shallow learning which requires domain expertise to hand design feature extractors, deep learning can learn good features automatically and the learning procedure can be easily generalized to solve other problems. In this article, deep learning method is applied to solve two different problems that related to domestic robot. First, a state-of-the-art multiple non-linear layer network structure based on convolutional-recursive neural network is presented to solve the problem of 3D image recognition which helps robot to differentiate objects in household environment. Second, a non-linear generative mode based on conditional restricted boltzmann machine is presented to solve the problem of human motion modeling which helps robot to model human motion and make action anticipation.**

Keywords— **Deep Learning, Convolutional Neural Network, Recursive Neural Network, RGB-D image recognition, Restricted Boltzmann machine.**

I. Introduction

Deep Learning is a new area of Machine Learning research. It is at the intersection of neural networks, artificial intelligence and optimization. Deep learning algorithms use many layers with non-linear process to extract features and learn different level of abstraction of data. Deep Learning allows to represent in compact way functions highly variable and highly non-linear. Compared with shallow learning that use hand design feature extractors, Deep Learning can learn good features from data in a non-supervised way, which does not requires a considerable amount of engineering skill or domain expertise. There are several kinds of Deep Learning algorithms. Most of them are inspired by neuroscience. Deep Learning is very useful for computer vision, speech recognition and in many other areas [1].

This paper illustrates how Deep Learning algorithms could be used for robotic aims. For example, some research teams' aim is to develop robotic technologies for personal domestic applications [2]. These robots need to be efficient in 3D object recognition, speech recognition and movement learning. We only focus on object recognition and movement learning.

Part II first explains the dataset used and a visualization of one example in the dataset is shown. Then theories about a kind of useful neural network structures for 3D objects recognition are explained, i.e., the neural network proposed by Richard Socher et al [3]. And a concrete view inside the structure is given to explain every processing techniques that are applied to 3D image recognition. Lastly, results of training a small scale version of Socher's network is presented.

Part III explains the Conditional Reduced Boltzmann Machines applied in motion generation.

II. 3D image recognition

The neural network analyzed in this section is the network proposed by Richard Socher et al [3]. The dataset is composed of RGB-D images of various sizes and of various objects which is provided by Lai et al.[4]. An RGB-D image is an image with four channels: three channels for colors (red, green, and blue = RGB) and one channel for the depth.

The dataset contains objects in 51 categories, each category has roughly 6 instances, resulting in 300 instances of these categories. And each object instance consists of multiple RGB-D frames that come from three video sequences, approximately

230 frames per sequence. Notice that, for every frame, the color information and depth information are saved separately in two files. The former is a three-channel unit8 RGB image while the latter is a single-channel unit16 depth image. Besides that, a corresponding object segmentation mask image is provided for every RGB-D frame. The dataset contains a total of 207,920 RGB-D frames.

One image instance is different from another, for example, the size of the first instance of apple is 78 ×84 pixels, and the second instance is 83×81 pixels, whereas the first instance of banana is 156×70 pixels. Notice that, for every instance, there are nearly 700 RGB images, not includes the corresponding segmentation mask and depth images. The size of the three images are the same. As shown in Fig 1.

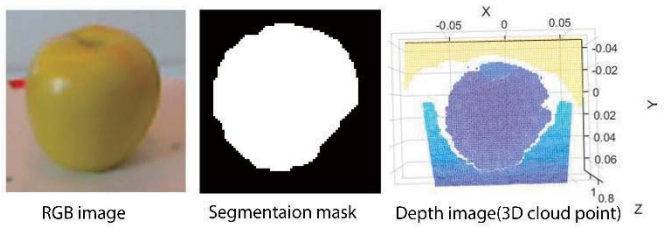

Fig 1 RGB, segmentation mask and depth images of one example of the third instance of apple.

However, before processing these images, they are resized to 148×148.

The aim is to obtain a network where data is the inputs of the network and the outputs give the object class.

The following three subsections will present the structure of Socher's network. The first subsection gives an overview the network structure and focus on the theories and different techniques used. The second subsection gives a concrete view of the network and focus on detailed procedures applied. The third subsection presents the results obtained by a small scale version of Socher's network.

A. Theory about Socher's network structure

Fig 2 shows the structure of the Socher's network. RGB and depth are processed separately with the same procedures. The structure of a CNN (Convolutional Neural Network) layer, RNN (Recursive Neural Network) and softmax classifier are explained in following subsections.

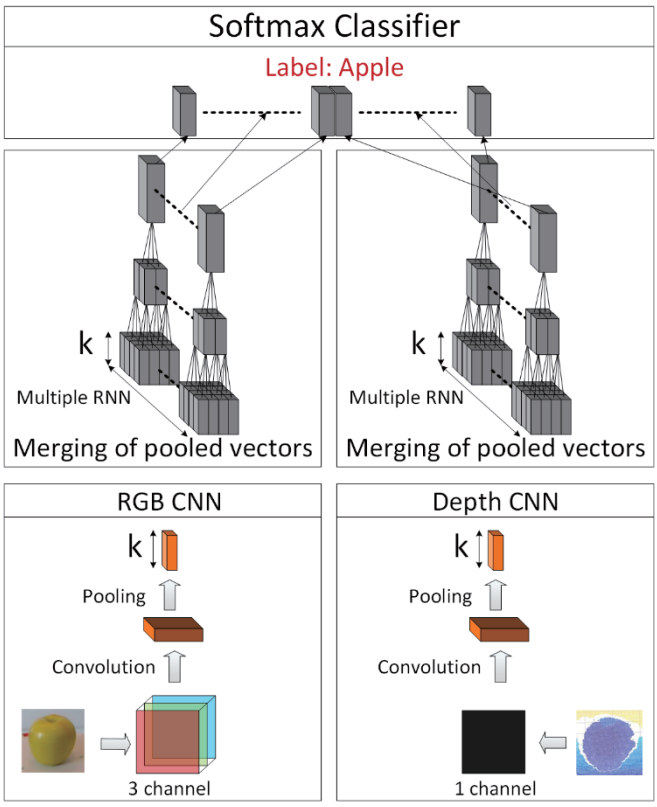

Fig 2 An overview of Socher's network: RGB and depth image are processed separately with a single CNN, the output of CNN are given as input to multiple RNNs. Each RNN has a fixed tree structure which can map input into a vector. All the vectors concatenated are given as input features of a softmax classifier.

1) Convolutional Layer, pooling and recursive neural network

The first layer of the network is a convolutional layer. In this subsection, we explain what a convolutional layer is.

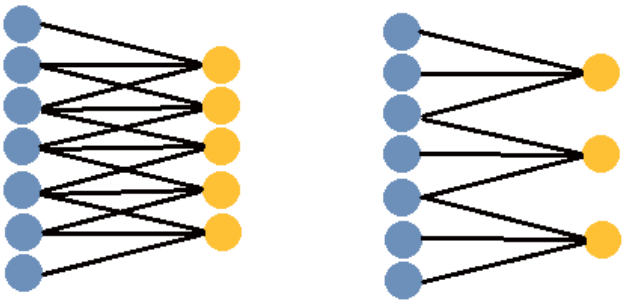

Fig 3 Two examples of convolutional layers with different stride size: stride size= 1 on left, stride size =2 on right

In Fig 3 blue neurons represent inputs of the convolutional layer and yellow neurons are outputs. These neurons aren't fully-connected. A yellow neuron is just connected with a part of the input neurons (three neurons in the example). The spatial extend of the neuron connectivity is called receptive field. The receptive fields of different neurons can overlap. For example in Fig 3, on the left the stride size is one and on the right the stride size is two.

In a convolutional layer, filters are convolved with the input neurons. Parameters learned are filters.

In the Socher's network, RGB channels and depth channel are convolved separately. Fig 4 illustrates the convolution principle. In this example, the filter dimension is 2x2. In this representation, neurons aren't arranged in rows but the idea is exactly the same as shown in Fig 43. Filter values represent connection weights. It's the same filter for the whole feature map. Therefore, weight values are shared by a lot of connections.

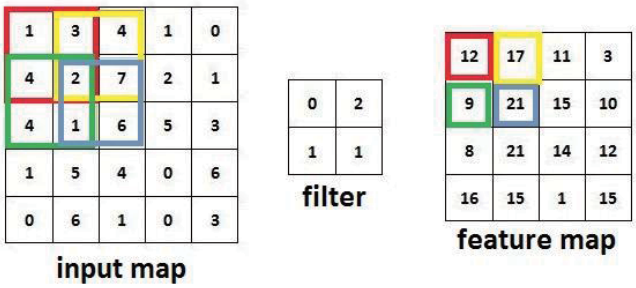

Fig 4 Convolution for a single channel : each neuron of the feature map is the dot product result of the filter and a part of the input map neurons. The stride size depends on parameters.

For RGB channels, filters are learned in three dimensions.

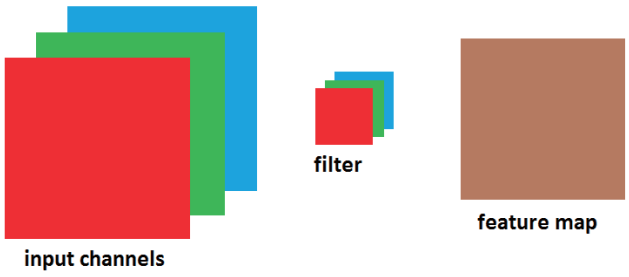

Fig 5 Convolution for three input channels. The feature map is in two dimensions.

In a convolutional layer, the network learns k filters. So the output of the convolutional layer is k feature maps.

The filters initialization in Socher's network is detailed in subsection B.

This convolutional layer allows to find low-level features. Moreover, these features are transitionally invariant.

To clarify image dimensions, the input image dimension is d_i (all images are square). The filter dimension is d_f . Feature map dimension is $l = d_i - d_f + 1$. With K filters, we obtain an $l \times l \times K$ matrix.

Then, we average pool feature maps with square regions of size d_p and a stride size of s. The output of the convolutional layer is a $r \times r \times K$ matrix, where $r = \frac{d_i - d_p}{s+1}$.

This output is the input of the recursive neural network (RNN) [5] [6]. We are not going to explain what really a RNN is. But the principle is to apply the same neural network recursively. This allows to learn hierarchical feature representations. Fig 6 illustrates the difference in the structure of a RNN and a feedforward network.

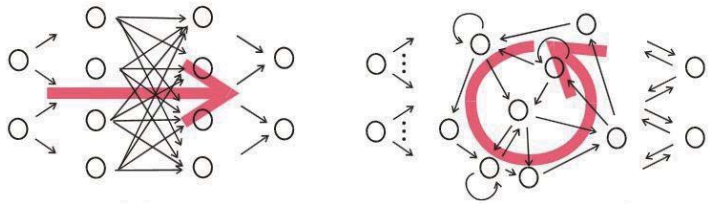

Fig 6 Typical structure of a feedforward network (left) and a recurrent network. Image source : [7]

2) Softmax classifier

To understand what is a softmax classifier, we start by explaining what is a regression. Let $(x^{(i)}, y^{(i)}) \in (\mathbb{R}^n, \mathbb{R}^n)$ be a family of vectors. The aim is to learn a function h such as $h(x^{(i)}) \approx y^{(i)}$.

If h is a linear function, then the regression is a linear regression [8]. To find h, we use a family of functions $h_\theta(x) = \sum_j \theta_j x_j = \theta^T x$. The goal is to find

a value of θ so that $h_\theta(x^{(i)})$ is closest as possible to $y^{(i)}$. To this end, a cost function is used: $J(\theta)=\frac{1}{2}\sum_i(h_\theta(x^{(i)})-y^{(i)})^2$. We choose θ which minimizes J(θ), using a gradient descent.

We also can use regression to classify $x^{(i)}$ between two classes. So $x^{(i)}$ go with $y^{(i)}\in\{0,1\}$. If $y^{(i)}$=0, then $x^{(i)}$ belongs to the first class ("0" class) and if $y^{(i)}$=1, then $x^{(i)}$ belongs to the second class ("1" class). In this case, a linear regression isn't appropriate. To find *h* we use a family of functions $h_\theta(x)=\frac{1}{1+\exp(-\theta^T x)}$. We search a value of θ so that $h_\theta(x)$ is large when *x* belongs to the "1" class and is small when *x* belongs to the "0" class. The cost function used is:

$$J(\theta)=-\sum_i(y^{(i)}\log(h_\theta(x^{(i)}))+(1-y^{(i)})\log(1-h_\theta(x^{(i)})))$$

This is a logistic regression [9].

A softmax regression is a generalization of logistic regression [10]. In this case, $x^{(i)}$ belongs to K classes and $y^{(i)}\in\{1,2,...,K\}$. The parameters of the model are $\theta^{(1)},\theta^{(2)},...,\theta^{(K)}\in\mathbb{R}^n$. θ is therefore a n-by-K matrix, $\theta=\left[\theta^{(1)},\theta^{(2)},...,\theta^{(K)}\right]$. The family of functions used is:

$$h_\theta(x)=\frac{1}{\sum_{j=1}^{K}\exp(\theta^{(j)T}x)}\begin{bmatrix}\exp(\theta^{(1)T}x)\\ \exp(\theta^{(2)T}x)\\ \cdots\\ \exp(\theta^{(K)T}x)\end{bmatrix}.$$

The cost function is:

$$J(\theta)=-\left[\sum_i\sum_{k=1}^{K}1\{y^{(i)}=k\}\log\frac{\exp(\theta^{(k)T}x^{(i)})}{\sum_{j=1}^{K}\exp(\theta^{(j)T}x^{(i)})}\right]$$

where $1\{y^{(i)}=k\}=1$ when $y^{(i)}=k$ and $1\{y^{(i)}=k\}=0$ when $y^{(i)}\neq k$.

Let *h* be the function learnt. Then $h(x)\in\mathbb{R}^K$, $h(x)=\begin{bmatrix}h^1(x)\\ h^2(x)\\ \cdots\\ h^K(x)\end{bmatrix}$. If $h^J(x)$ is the highest value of all $h^i(x)$ then *x* belongs to the class *J*.

B. Details about processing

As we know, RGB image has three channels, while depth image only has one. Firstly, for each channel of resized image, a huge image patches is obtained by applying a 9×9 sliding window in each image channel. Given the stride size of sliding is one, 19,600 image patches are obtained. Fig 7 shows the case about RGB image processing, the huge image patches is saved by a matrix of 234×196,000, each column denotes one possible image patches extracted by the 9×9 sliding window. Then a number of columns of the huge patches are randomly picked, the number of columns chosen depends on information extracted from the corresponding segmentation mask. In the example shown Fig 7, 63 columns are chosen.

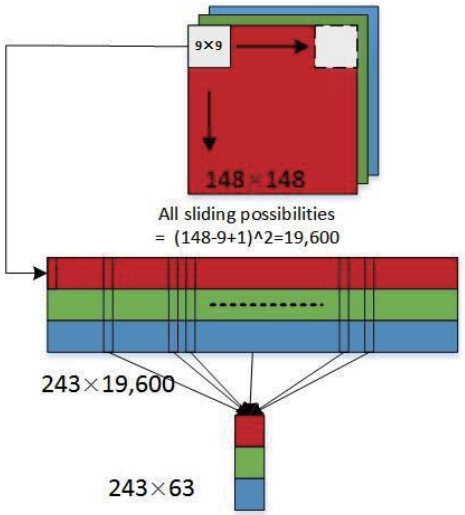

Fig 7 Pretrain procedure of getting all possible patches from one images, and random sampling from the patches

The chosen columns from other images aggregate an even greater patches, in Socher's network the number of extracted patches is 500,000, as shown in Fig 8. After that, k-means is performed

on this aggregation of patches and results in 128 centroids.

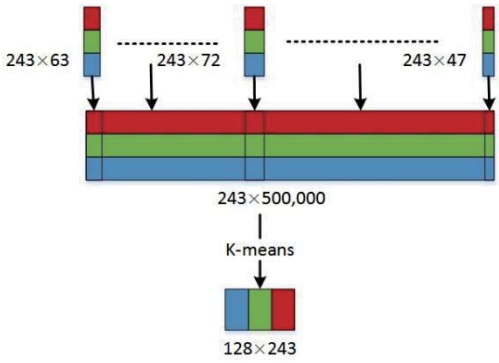

Fig 8 Pretrain procedure of getting CNN filters from aggreation of image patches by k-means

Each centroid is a 243 dimension vector, these vectors are then used as filters for CNN. If these vectors are rearranged as 9×9×3, we can have a visualization of these RGB filters, for depth filters, vectors need to be rearranged as 9×9. Fig 9 shows both the depth and RGB filters which are learned from non-supervised learning process.

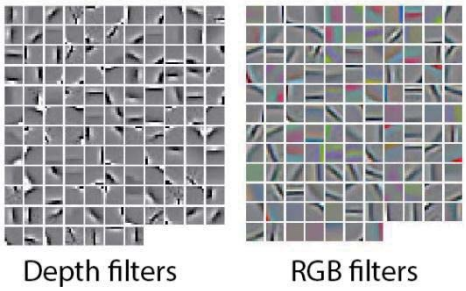

Fig 9 Filters used in CNN which are learned by non-supervised learning using k-means (learned from 40,000 images patches)

Secondly, for the RGB-D dataset, a split table is used to split the whole dataset as 80% training dataset and 20% testing dataset. Then training dataset is used in the forward propagation of the single CNN layer.

Fig 4 explains the principle idea of convolution, in Socher's network, the input map is the RGB image or depth image. If the input map is the RGB image, then it is represented by a 148×148×3 matrix, the correspond filter size is 9×9×3, thus, each filter convolves with the RGB image will result a 140×140 feature map, if all the 128 filters are applied, the feature map will be a matrix of 128×140×140. The feature map is then rectified with absolute value, local normalized and average pooled. After pooling, the feature map turned out to be a matrix of 128×27×27 which is used as input of multiple RNNs.

Inspired by application of RNN in natural langue processing, the tree structure generated when process a sentence is replaced by a fixed tree structure with randomly initialized weights.

Fig 2 shows the tree structure of RNN, however the reception field actually used is 3×3 instead of 2×2 which is shown in Fig 2. The original input matrix of each RNN is the output of the pooled convolution as described above, this leads to the following matrices at each depth of the tree: 128×27×27 to 128×9×9 to 128×3×3 to finally 128. Then 128 randomly initialized RNNs are applied both in the processing of RGB and depth. Each RNN will output a 128 dimensional vector, after forward propagate through all the 256 RNNs, the final features will have 256×128 = 32,768 dimensions. Finally, the final features are used to train a softmax classifier.

C. Results obtained by training a small scale version of Socher's network

Code matlab for training and testing Socher's network is available at socher.org. However, the code needs to be modified in order to obtain the value of filters and the weights of the trained softmax classifier. 500,000 extracted image patches are used to get filters for CNN, the output of CNN used as input of 128 RNNs, the concatenated features which has a dimension of 32,768 are used to train the softmax classifier on approximately 33,000 training examples.

The amount of computing and space of RAM memories required to train the network is very high, which makes a server equipped with intel xeon E5 processor and 128GB RAM to run weeks to full training and testing the network. When this article is written, we have successfully trained a half of the network, i.e., the RGB modality.

Instead of waiting the result of the fully trained network, a small scale version of Socher's network

is trained and analyzed. The number of image patches is set to be 40,000 instead of 500,000. The number of RNNs used also decreased from 128 to 64, hence the final features is a 16,384 dimensional vector, the softmax classifier is trained on 2,535 training examples to differentiate 18 categories, finally 485 testing examples from corresponding categories is randomly chosen to test the estimation result of the network.

The percent corrected of RGB modality is 82.6263%, the percent corrected of depth modality is 89.4949%. However, when features from RGB and depth modality is combined, the percent corrected decreased to 78.1818%. The result shows that the network is overfitting. Because the concatenated features are too many to differentiate only 18 categories.

III. MODELING HUMAN MOTION

The neural network analyzed in this section is the network proposed by GrahamW. Taylor, Geoffrey E. Hinton and Sam Roweis [11]. This is a non-linear model, able to generate human motion. The data given in input is motion frames, described by the space coordinates of body elements (knees, elbows, feet, shoulders, hands, torso), which were recorded in a motion picture lab.

Fig 10 Example (in 2D) of data used in the training set. Image source: [12]

The dataset is composed by several records of a predefined type of motion (walking, running, sitting, and alternation between walking and running) taken from the CMU Graphics Lab Motion Capture Database or E. Hsu, K. Pulli, and J. Popoviê database. The both datasets were first recorded at 120 frames per second, and then down sampled to 30 [11].

The aim of this network is, given an output of three successive motion frames, to successfully generate the following frame.

A. Conditional Reduced Boltzmann Machine

The network used is a combination of several Conditional Reduced Boltzmann Machines (CRBM). In order to fully understand the particularities of this kind of network, we will describe the process of design of such a network, by defining what are Reduced Boltzmann Machines and Deep Belief Networks (DBN). After acknowledging this, we will discuss the learning algorithm.

1) Network structure

Boltzmann Machines (BM) are designed on the works of Boltzmann in statistical mechanics. A simple application of this theory is the harmonization of energy between molecules in a closed environment, leading to a stable global state, even if some elements were exited at first. For example, heated gazes cool when their molecules spare their energy by hitting each other. BMs work the same. They are composed of two layers: the visible layer which receive the inputs, and the hidden layer. For a given input, which we can read as an amount on energy, is given to the visible layer, every unit perform energy transfers to the other until reaching a stable state. This means that the links between the all the units are bidirectional: all can communicate or receive energy. The idea is the same in a RBM, but there is no connection between elements of the same layer.

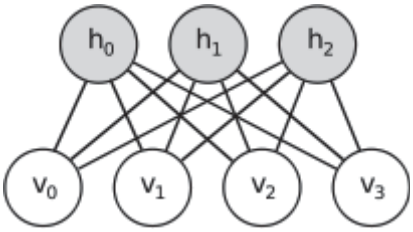

Fig 11 Reduced Boltzmann Machine. Image source: [13]

Such networks are highly used in unsupervised learning, because they are able to discover by themselves abstract concept.

A DBN is a network composed by a visible layer, and several hidden layers. The top two hidden layers of a DBN are a RBM. In order to understand the structure of a DBN, we can easily compare (Fig 412) it to a Deep Boltzmann Machine (DBM), a structure built on an addition of RBMs. Indeed, the main difference between the two is that, in contrary to a DBM, a DBN uses two types of networks. The top is effectively a RBM, but the other layers are composed by units using only unidirectional links. Usually, we prefer using structure like DBNs, because adding RBMs tends to a tremendous amount of computation.

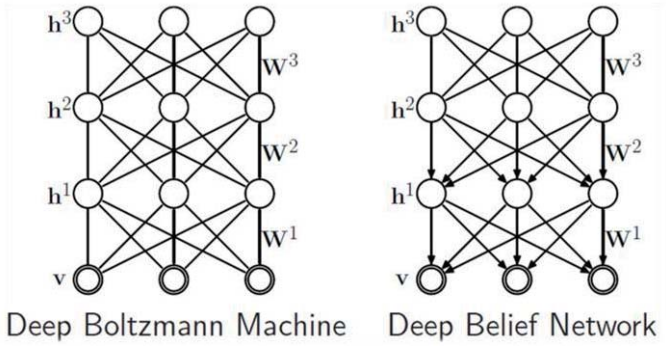

Fig 12 DBM and DBN. Image source: [14]

A CRBM is very similar to a RBM, but the major difference is that there are links between the elements of the visible layer.

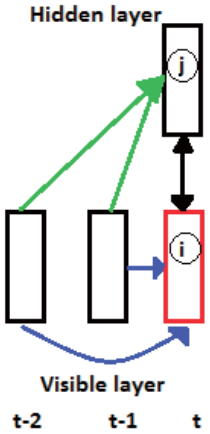

Fig 13 Conditional Reduced Boltzmann Machine. Image source: [11]

These links in the visible layer help modeling the temporal dependencies of the successive frames. Indeed, to be able to generate motion frames, it is necessary to know the related previous events in order to predict what could happen. So, as we can see in Fig 13, the "t-2" and "t-1" units are linked to the "current time" or "t" units. This structure can be used in a DBN, we will see next how to build and train such network.

2) Training and building

Due to its mechanism, the learning phase of a RBM puts forward computational considerations. As shown in Fig 14, we should compute energy transfer until reaching convergence in order to figure out the weights. But we cannot compute an infinite amount of energy transfer as it is the case in physics.

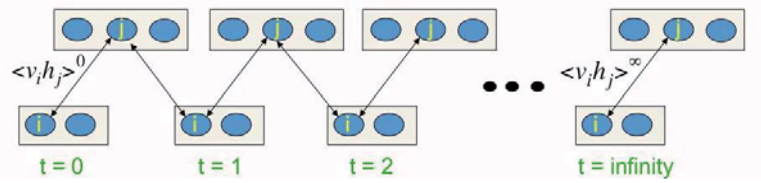

Fig 14 Theoretical convergence of a RBM

In order to solve this dilemma, we use a contrastive divergence algorithm [16]. Instead of calculating a huge amount of energy transfer, we only do it for two phases as shown in Fig 15. The first is the "positive phase" where we compute contributions from directed visible-to-hidden connections, represented by $\langle v_i h_j \rangle^0$ in Fig 15, the contribution of the i^{th} unit of the visible layer to the j^{th} unit of the hidden layer. The second one is the "negative phase" where we compute posterior probability for hidden units represented by $\langle v_i h_j \rangle^1$ in Fig 15, where we reconstruct the visible layer in order to compute this post probability.

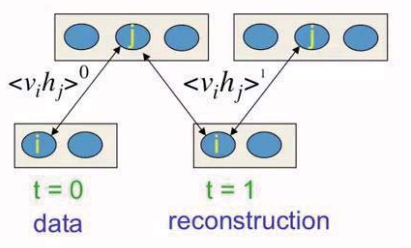

Fig 15 Positive and negative phase of contrastive divergence

So, for each data given to the visible layer, the weights undergo a variation of :

$$\Delta w_{ij} = \varepsilon \left(\langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^1 \right),$$

where ε is the learning rate.

In order to design a DBN, we often use a greedy layer-wise unsupervised algorithm [15]. The main idea is to first build a DBN with only one hidden layer, which is a simple RBM and to train it with the contrastive divergence algorithm. Then, we add another hidden layer of RBM at the top of the network and use the previously computed weights with contrastive divergence to initialize the weights of the other layers, to finally repeat the learning process to the top layer.

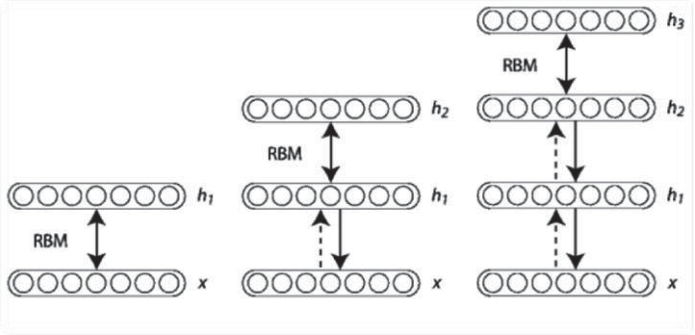

Fig 16 Building a DBN with contrastive divergence. Image source: [14]

This algorithm also works with CRBM, but we only keep in the end the "t" units in the hidden layers, since they are not useful when we use this network as a DBN.

B. Results

In order to generate a walking motion, we used three different movements of 438, 260 and 3128 frames. The frames are mixed in a larger data batch, so that the network does not learn on the dataset in the time order of the frames, and thus create a better training database. In order to generate motion frames, we initialize the visible layer with three successive frames. The results are shown on a 3D graph, where the successive frames are animated:

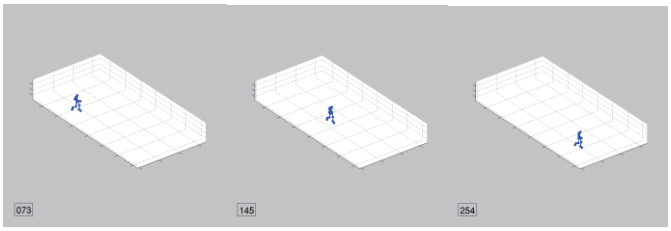

Fig 17 Motion frames generated, animated in 3D in Matlab.

It is also possible to visualize them in 2D:

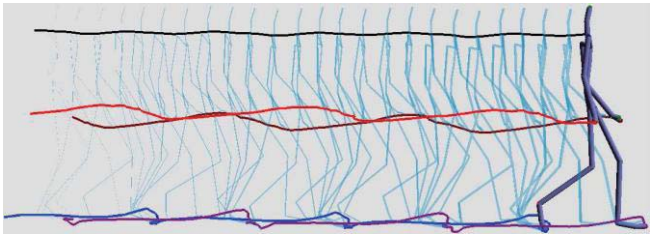

Fig 18 Motion frames generated in 2D. Image source: [11]

IV. CONCLUSION

In this paper we present two algorithms, one for 3D image recognition and one for modeling human motion. These algorithms could be used for robot tasks.

CNN is very specific for image analysis. With its filters, it supposes that there is a correlation between the inputs. Indeed in an image, a pixel is very linked with its neighbors. They are not completely independent. DBN supposes nothing about the inputs. It finds itself interesting features. DBN is also efficient for image recognition, but less than CNN. However, CNN is not really efficient for modeling human motion.

REFERENCES

[1] S. M. Metev and V. P. Veiko, *Laser Assisted Microtechnology*, 2nd ed., R. M. Osgood, Jr., Ed. Berlin, Germany: Springer-Verlag, 1998.

[2] Loy van Beek, Kai Chen, Dirk Holz, Mauricio Matamoros, Caleb Rascon, Maja Rudinac, Javier Ruiz des Solar, Sven Wachsmuth, *"Robocup@Home 2015:Rule and regulations"*, http://www.robocupathome.org/rules/2015_rulebook.pd, 2015

[3] Richard Socher, Brody Huval, Bharath Bhat, Christopher D. Manning, Andrew Y. Ng, *" Convolutional-Recursive Deep Learning for 3D Object Classification"* , 2012

[4] Kevin Lai, Liefeng Bo, Xiaofeng Ren, and Dieter Fox. *A Large-Scale Hierarchical Multi-View RGB-D Object Dataset*. In IEEE International Conference on Robotics and Automation (ICRA), May 2011.

[5] Richard Socher, C.Lin, Andrew Y.Ng, and Christopher D. Manning, *Parsing Natural Scenes and Natural Language with Recursive Neural Networks*. In ICML, 2011

[6] J. B. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46, 1990.

[7] H. Jaeger. *Tutorial on training recurrent neural network, covering BPPT, RTRL, EKF and the "echo state network" approach.* GMD Report 159, German National Research Center for Information Technology, 2002 (48pp.)
[8] UFLDL Tutorial, Linear Regression [Online]. Available : http://ufldl.stanford.edu/tutorial/supervised/LinearRegression/
[9] UFLDL Tutorial, Logistic Regression [Online]. Available : http://ufldl.stanford.edu/tutorial/supervised/LogisticRegression/
[10] UFLDL Tutorial, Softmax Regression [Online]. Available : http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/
[11] GrahamW. Taylor, Geoffrey E. Hinton and Sam Roweis. *Modeling Human Motion Using Binary Latent Variables*, 2006.
[12] E. Hsu, K. Pulli, and J. Popoviê, "Style translation for human motion," *ACM Trans. Graph*., vol. 24, no. 3, pp. 1082–1089, 2005.
[13] Deeplearning.net, Restricted Boltzmann Machines [Online]. Available: http://deeplearning.net/tutorial/rbm.html
[14] Deeplearning.net, Deep Belief Network [Online]. Available: http://deeplearning.net/tutorial/DBN.html
[15] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochell, *Greedy Layer-Wise Training of Deep Networks*, 2007.
[16] G.E.Hinton "Training products of experts by minimizing contrastive divergence, *"Neural Comput"*, vol.14,pp.1771-1800, Aug 2002.