PRIVALOV V - USING DEEP LEARNING FOR OBGECT RECOGNITION IN ROBOTICS

Motivation and aims

My research project is related to the object recognition task in service robotics.

Object recognition is the most important ability of robot. The ability to identify and differentiate objects is the basis of the intelligence of robots which must move in an environment and interact with it.

Object recognition in robotics is a quite expensive task because it involves complex intelligent analysis of images. Image analysis requires much of computational resources that are critically limited on robotic platforms.

Deep learning is a focus of the current artificial intelligence research because of the advantage of self-teaching which reduce manual labour.

We are intended to research if Deep learning is capable to solve the object recognition task more effectively than standard methods and it will reduce manual labor.

Definition of DL

Deep learning is a new concept in machine learning. It allows representation learning, i.e. learning of feature representations automatically instead of having to define them manually based on expert knowledge.

Research in this area attempts to find better representations of data (e.g. image) that make it easier to learn tasks of interest (e.g., is this the image of a human face?) from examples. One of basic deep learning architectures is deep neural networks.

History of DL

Let’s take a look on history of Deep learning in context of neural networks. There is represented various phases of development deep learning in the hype cycle. The peak activities (“expectations” or “media hype” on the vertical axis) occurred in late 1980s and early 1990s. Kunihiko Fukushima in 1980 introduced first Deep learning architecture - Neocognitron. In 1989, Yann LeCun was able to apply the standard backpropagation algorithm to a deep neural network with the purpose of recognizing handwritten ZIP codes on mail. The time to train the network on this dataset was approximately 3 days, making it impractical for general use. Because of speed issues simpler models such as support vector machines (SVMs) became the popular choice of the field in the 1990s and 2000s.

When new deep neural architectures appeared in 2009, the learning became highly effective and this has inspired the subsequent fast growing research (“enlightenment” phase shown in the slide). The height of the “plateau of productivity” phase, not yet reached, it will grow further.

The term "deep learning" became commonly used in the mid-2000s after a publication by Geoffrey Hinton and Ruslan Salakhutdinov showed how a many-layered feedforward neural network could be effectively pre-trained one layer at a time, using supervised backpropagation for fine-tuning. The deep belief network (DBN) and a fast algorithm for training it were invented in 2006. Industry-scale application of DBN and DNN for speech recognition started in 2009.

Current results of using DL

With this greater depth, NN are producing remarkable advances in speech and image recognition. June 2012, a Google used deep-learning system for recognition objects like cats in images from YouTube videos.

Certainly machine intelligence is starting to transform everything from communications and computing to medicine, manufacturing, and transportation.

The IBM's Watson computer uses some deep-learning techniques and is now being trained to help doctors make better decisions. Microsoft has deployed deep learning in its Windows Phone and Bing voice search.

But for now, says Peter Lee, head of Microsoft Research USA, "deep learning has reignited some of the grand challenges in artificial intelligence."

Application of Deep learning in robotics

Most of the research in the field of deep learning for robotics are concerning the recognition of objects. Deep learning was already successfully used in the method of detecting robotic grasps for novel objects that allows to avoid hand design of features. There Deep learning was applied to find the best grasp pose - a 2D oriented rectangle in image space, with two edges corresponding to the gripper

plates. Also deep learning methods are used in modeling of robot behavior based on training demonstrations.

Definition of neural network

Neural networks are nonlinear models motivated by the physiological architecture of the nervous system. They involve a set of simple nonlinear computations that when aggregated can implement robust and complex nonlinear functions. The general idea behind neural networks is pretty straightforward: map some input onto a desired target value using a distributed set of nonlinear transformations.

Neural networks can have any number of layers. Networks with one hidden layer is called shallow, with many – deep.

Since 2006, a set of techniques has been developed that enable learning in deep neural nets. These techniques have enabled much deeper (and larger) networks to be trained - people now routinely train networks with 5 to 10 hidden layers. And, it turns out that these perform far better on many problems than shallow neural networks. The reason is the ability of deep nets to build up a complex hierarchy of concepts.

So neural networks can compute any function but deep networks are the networks best adapted to learn the functions useful in solving many real-world problems.

Perceptrons

The basic computation of NN is called perceptron. Perceptron calculates weighted sum of observed inputs $a = (a_1, a_2, \dots, a_N)$ multiplied by corresponding weights $w = (w_1, w_2, \dots, w_N)$ as pre-activation z. Oftentimes the preactivation calculation involves a bias *b* with value +1 that is added to an input. The pre-activation is transformed then by a nonlinear activation function *g(z)* to output a final activation a_{out}.

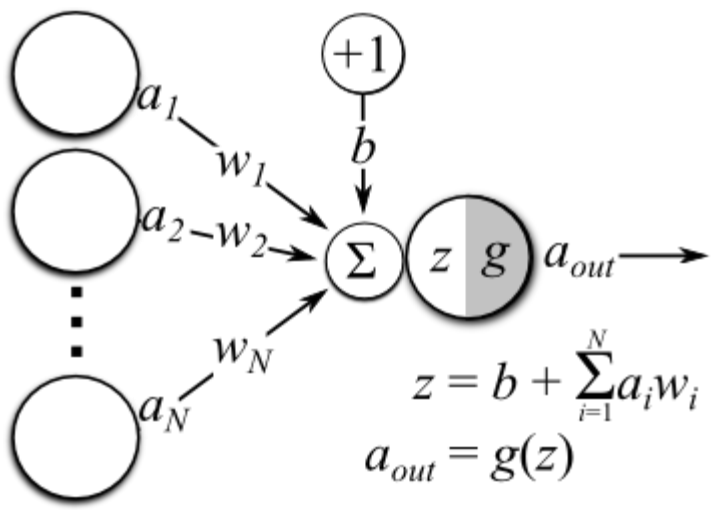

Figure 1: Diagram of neural network with one perceptron.

There are many types of the activation function $g(z)$, and the choice generally depends on the task we would like the network to perform. The most popular activation function is the logistic sigmoid:

$$g_{sigm}(z) = \frac{1}{1+e^{-z}},$$

which output values $g_{sigm} \in (0,1)$. When the network outputs use the logistic sigmoid activation function, the network implements linear binary classification. Third type of activation function - hyperbolic tangent function, *tahn(z)*, which outputs values $g_{tahn} \in (-1,1)$ - can also implement binary classification.

A key property of these activation functions is that they are all smooth and differentiable. Differentiability of activation function is important for training neural networks. The derivatives for each of these common activation functions are given here:

$$g'_{linear}(z) = 1$$

$$g'_{logistic}(z) = g_{logistic}(z)(1 - g_{logistic}(z))$$

$$g'_{tahn}(z) = 1 - g^2_{tahn}(z)$$

What is interesting about these derivatives is that they are either a constant (i.e. 1), or can be defined in terms of the original function. This makes them extremely convenient for efficiently training neural networks.

Feedforward Neural Networks

Feedforward neural network consists of more than one layers of perceptrons. Let's introduce a network with one input layer, one output layer and one layer between them. Intermediate layer is often referred to as a "hidden" layer, as it doesn't directly observe input or directly compute the output. By using a hidden layer, we form a multilayered neural network. For example, the network in the next slide (Figure 2) would be considered a 2-layer neural network because it has two layers of weights: those connecting the inputs to the hidden layer (w_{ij}), and those connecting the output of the hidden layer to the output layer (w_{jk}).

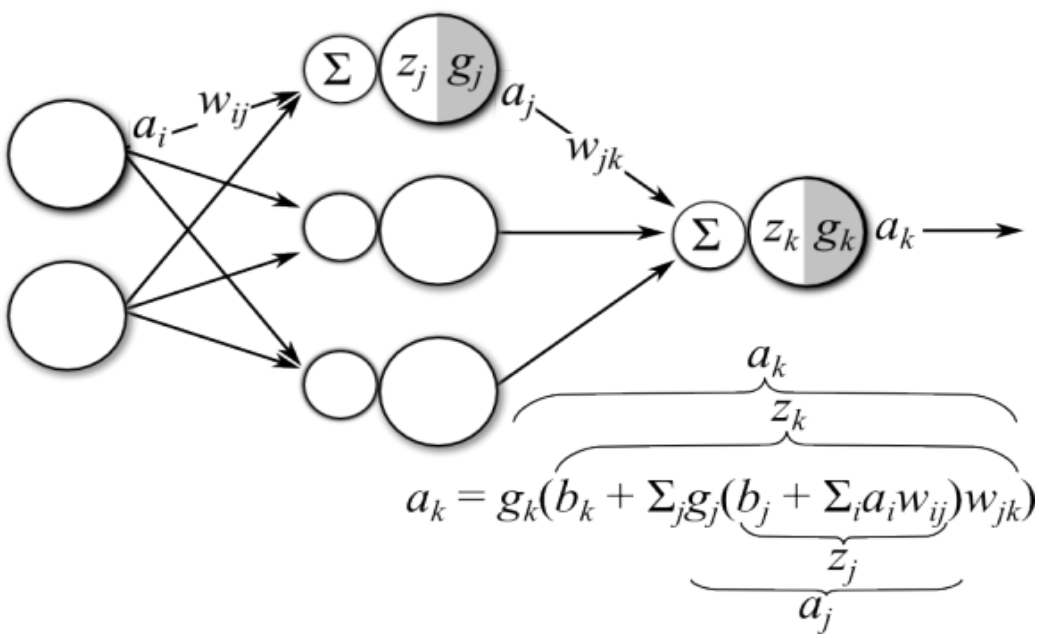

Figure 2: Diagram of a multi-layer ANN. Each node in the network can be considered a single-layered ANN (for simplicity, biases are not visualized in graphical model).

Let's denote input layer by *l*, *m* – number of nodes in layer *l* hidden layer is *l* + 1, *p* – number of nodes in layer *l* + 1.

Multi-layer neural networks form compositional functions that map the inputs nonlinearly to outputs. If we associate index i with the input layer, index j with the hidden layer, and index k with the output layer, then an output unit in that network computes an output value a_k given and input a_i via the following compositional function:

$$a_{out} = a_k = g_k(b_k + \sum_j g_j(b_j + \sum_i a_i w_{ij})w_{jk})$$

Training neural networks and gradient descent

Training neural networks involves determining the network parameters $\theta = \{W, b\}$ that minimize the errors that the network makes. We will define error function as squared difference between the network output and the target value:

$$E = \frac{1}{2}(output - target)^2$$

(Note that the squared error is not chosen arbitrarily, but has a number of theoretical benefits and considerations.) Having an error function, we then aim to find the setting of parameters that minimizes this error function. This concept can be interpreted spatially by imagining a "parameter space" (Figure 3) whose dimensions are the values of each of the model parameters, and for which the error function will form a surface of varying height depending on its value for each parameter. Model training is thus equivalent to finding point in parameter space that makes the height of the error surface small.

The task of training the parameters of an neural network generally is solved using gradient descent: the gradient descent algorithm first calculates the gradient

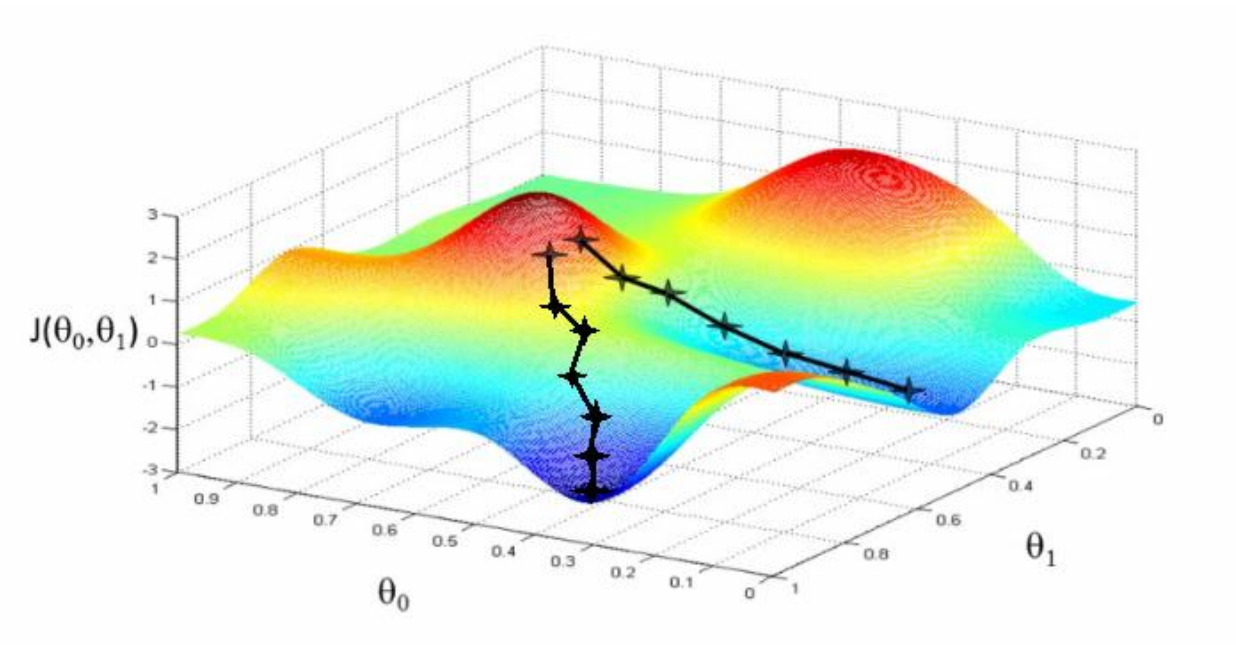

Figure 3: Graphical interpretation of learning task in “parameter space”.

$gradE = \frac{\partial E}{\partial \theta}$ of the error function with respect to each parameter θ of the model parameters . This gradient information will give us the direction in parameter space that decreases the height of the error surface. We then take a step in that direction and repeat, iteratively calculating the gradient and taking steps in parameter space.

The backpropagation algorithm

It turns out that the gradient information for the neural network error surface can be calculated efficiently using a message passing algorithm known as the backpropagation algorithm.

The main concept underlying the algorithm is that for a given observation we want to determine the degree of “responsibility” that each network parameter has for mis-predicting a target value associated with the observation. We then change that parameter according to this responsibility so that it reduces the network error.

These are presented main steps of the algorithm.

In order to determine the network error, we first propagate the observed input forward through the network layers. This is Step I of the backpropagation algorithm, and is demonstrated on the slide (Figure 4). Note that in the slide each of the network outputs is designated by a_k. Also note that when implementing this forward-propagation step, we should keep track of the feed-forward pre-activations z_l and activations a_l for all layers *l*, as these will be used for calculating backpropagated errors and error function gradients.

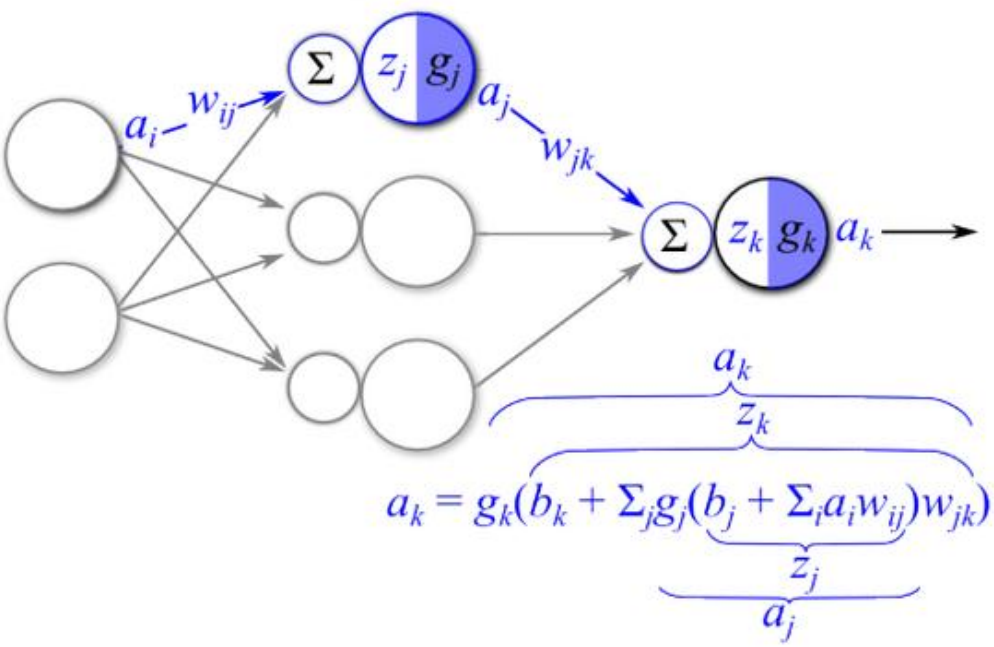

Figure 4: Step 1 of backpropagation – forward propagation of signal.

Step II of the algorithm is to calculate the network output error and backpropagate it toward the input. Let's again that we are using the sum of squared differences error function:

$$\mathrm{E} = \frac{1}{2}\sum_{\mathrm{k}\in\mathrm{K}}(\mathrm{a_k} - \mathrm{t_k})^2,$$

where we sum over the values of all *k* output units (one in this example). We can use error from node defined earlier as an "error signal" δ_k at the output node that will be backpropagated toward the input. The error signal is calculated as follows:

$$\delta_\mathrm{k} = \mathrm{g'}_\mathrm{k}(\mathrm{z_k})\mathrm{E'}(\mathrm{a_k}, \mathrm{t_k})$$

$$= g'_k(z_k)(a_k - t_k).$$

Thus the error signal essentially weights the gradient of the error function by the gradient of the output activation function (notice there is a z_k term is used in this calculation, which is why we keep it around during the forward-propagation step). We can continue backpropagating the error signal toward the input by passing δ_k through the output layer weights w_{jk}, summing over all output nodes, and passing the result through the gradient of the activation function at the hidden layer $g'_j(z_j)$ (Figure 5). Performing these operations results in the back-propagated error signal for the hidden layer, δ_j:

$$\delta_j = g'_j(z_j) \sum_k \delta_k w_{jk}.$$

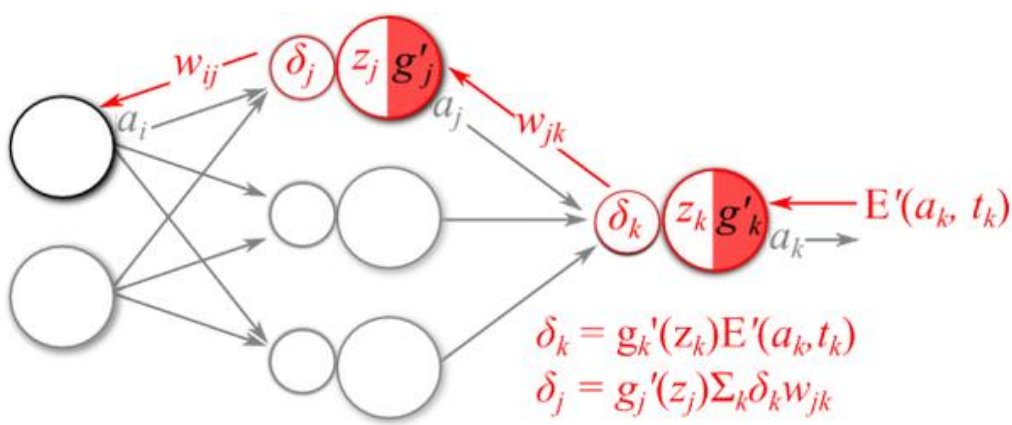

Figure 5: Step 2 of backpropagation – back propagation of signal.

For networks that have more than one hidden layer, this error backpropagation procedure can continue for layers $j - 1$, $j - 2$, …, etc.

Step III of the backpropagation algorithm is to calculate the gradients of the error function with respect to the model parameters at each layer l using the forward signals a_{l-1}, and the backward error signals δ_l. If one considers the model weights $w_{l-1,l}$ at a layer l as linking the forward signal a_{l-1} to the error signal δ_l (Figure 6), then the gradient of the error function with respect to those weights is:

$$\frac{\partial E}{\partial w_{l-1,l}} = a_{l-1}\delta_l$$

Note that this result is closely related to the concept of Hebbian learning in neuroscience. Thus the gradient of the error function with respect to the model weight at each layer can be efficiently calculated by simply keeping track of the forward-propagated activations feeding into that layer from below, and weighting those activations by the backward-propagated error signals feeding into that layer from above.

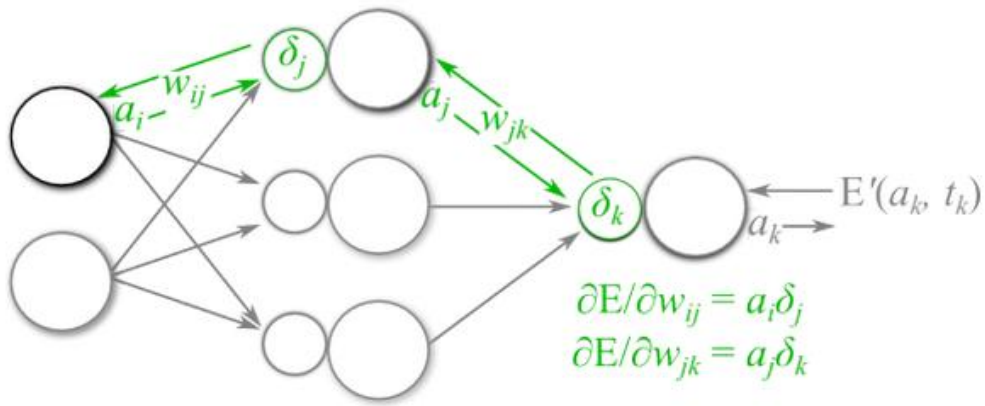

Figure 6: Step 3 of backpropagation – calculation parameter gradients.

What about the bias parameters? It turns out that the same gradient rule used for the weight weights applies, except that "feed-forward activations" for biases are always +1 (see Figure 1). Thus the bias gradients for layer *l* are simply:

$$\frac{\partial E}{\partial b_l} = (1)\delta_l = \delta_l$$

The fourth and final step of the backpropagation algorithm is to update the model parameters based on the gradients calculated in Step III. Note that the gradients point in the direction in parameter space that will increase the value of the error function. Thus when updating the model parameters we should choose to go in the opposite direction. How far do we travel in that direction? That is generally determined by a user-defined step size (aka learning rate) parameter, η.

Thus given the parameter gradients and the step size, the weights and biases for a given layer are updated accordingly:

$$\mathrm{w}_{l-1,l} \leftarrow \mathrm{w}_{l-1,l} - \eta \frac{\partial \mathrm{E}}{\partial w_{l-1,l}}$$

$$b_l \leftarrow b_l - \eta \frac{\partial \mathrm{E}}{\partial b_l}.$$

To train an ANN, the four steps outlined above and in Figures 4-6 are repeated iteratively by observing many input-target pairs and updating the parameters until either the network error reaches a tolerably low value or a set number of parameter updates has been achieved.

There is an optimized type of gradient descent - stochastic gradient descent. The idea is to estimate the gradient $gradE$ by computing $gradE_x$ for a small sample of randomly chosen training inputs. By averaging over this small sample it turns out that we can quickly get a good estimate of the true gradient $gradE$, and this helps speed up gradient descent, and thus learning.

To make these ideas more precise, stochastic gradient descent works by picking out a small number *m* of randomly chosen training inputs. Then stochastic gradient descent works by picking out a randomly chosen mini-batch of training inputs, and training with those,

$$\mathrm{w_k} \leftarrow \mathrm{w_k} - \frac{\eta}{\mathrm{m}} \sum_{\mathrm{x}} \frac{\partial \mathrm{E_x}}{\partial \mathrm{w_k}}$$

$$\mathrm{b_l} \leftarrow \mathrm{b_l} - \frac{\eta}{\mathrm{m}} \sum_{\mathrm{x}} \frac{\partial \mathrm{E_x}}{\partial \mathrm{b_l}},$$

where the sums are over all the training examples E_x in the current mini-batch. Then we pick out another randomly chosen mini-batch and train with those. And so on, until we've exhausted the training inputs, which is said to complete an *epoch* of training. At that point we start over with a new training epoch.

Convolutional neural networks

A Convolutional Neural Network (CNN) consists of two types of layers: convolutional layers and fully connected layers as in a standard multilayer neural network. The architecture of a CNN is designed to take advantage of the 2D structure of an input (image, speech signal).

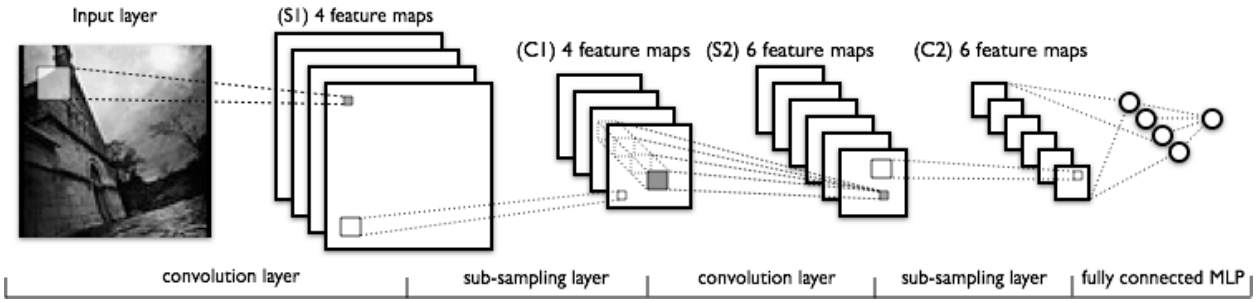

Figure 7: Image representation in convolutional neural network for object recognition task.

In this example (Figure 7) convolutional network is used for object recognition in an image. First layers – convolutional – are used for retrieving features from image. The input to the first fully-connected layer is the set of all features maps at the layer below.

Advantageous of DL

The most important advantageous of Deep learning are follow:

1. Hierarchical representation of data is effective way of representation because world is compositional (images have a lot of local structure, complex images are formed by combinations of small parts (edges, gradients) that are represented by some random combination of pixels). When attacking problems such as image recognition, it helps to use a system that understands not just individual pixels, but also increasingly more complex concepts: from edges to simple geometric shapes, all the way up through

complex, multi-object scenes. Hierarchical representation allows to learn things level-by-level from mimimum elements like image pixels. We can learn multiple levels of representation.

2. Deep architectures can represent some classes of functions more compactly, and therefore better performance on fewer examples should be expected.
3. Concept of Deep learning allows to configure and modify architecture of network in any desirable way (it's possible to change all parameters of network: weights, biases, nodes, connections between nodes).

Further plans

Our goal is to investigate abilities of Deep learning relating to the task of object recognition. We will apply different types of deep neural architectures to specific tasks and estimate their effectiveness. For practical applicability of Deep learning in robotics, a particular attention will be paid to efficient implementations of deep learning model on limited computing resources like the ones that are available on the robot.

Conclusion

So we made introduction to Deep learning, gave definition of that term, reviewed basics of neural networks as the main concept of Deep learning and algorithms used by DL, reviewed an example of deep neural networks and defined main advantageous of concept of Deep learning.

Deep learning has already shown itself as a quite effective technology in the areas of computer vision and robotics because of hierarchical structure of deep neural networks and their ability to represent any complex nonlinear function. This

technology was already successfully applied for solving essential tasks in service robotics and it could be applied to solving many other complex tasks.

The ability to customize deep networks is an important property that makes the deep networks so popular and usable in many spheres and makes possible emergence of new neural networks architectures.