Developmental Neural Learning for a Visual Concept Steering Control Task

Jiating Zhu

June 3, 2015

Acknowledgements

I would like to thank my supervisor, Dr Andrea Soltoggio, for his support and guidance throughout my time as his student. His enthusiasm, encouragement and teachings are priceless.

Abstract

A developmental neural learning method is proposed in this report. The problem analyzed consists in a learning process in which both low and high level concepts need to be progressively acquired. Low level concepts are average orientations of lines that can be observed in visual stimuli. Higher level concepts are control policies to be learned according to the average orientations of lines in the visual stimulus. The challenge is to devise an autonomous neural learning process that abstracts features from low to high level, and operates using both unsupervised and reinforcement learning paradigms. A three-layer network that learns visual concepts of directions is trained by means of a novel three-step procedure, which is also the main focus of the current study. The experimental results show that the agent can successfully identify the visual concepts of directions and react to the images of directions correctly. The accuracy of the direction identification is high when using the three-step training method, whereas the accuracy becomes worse when skipping any step in the proposed training method. The relationship and interplay of the three training steps and the principle of the training method are discussed. This result suggests that all three steps are essential to develop a correct network topology for this particular learning problem. In short, a procedure for autonomous and effective neural learning is proposed for a complex perception and control task.

Keywords: Developmental neural learning, Unsupervised learning, Reinforcement learning, Visual concept learning, Control task.

Contents

1 Introduction

1.1 Overall Aim

This project aims at providing an insight onto the dynamics of neural learning when the task is a combination of perception (image classification) and control policies (an agent performing steering actions). Although the network used has three layers, the type of learning can be applied to deeper structures for improve classification and control capabilities.

1.2 Background

Data representation is a key factor in the performance of machine learning. The inability to extract and manage the characteristic information from data is the major weakness of current learning algorithms [1]. Deep learning has its advantage in representation learning and has some success in both academia and industry, such as speech recognition and signal processing, object recognition, natural language processing, and multi-task, transfer learning and domain adaptation [1].

Deep learning methods are formed by the composition of multiple non-linear transformations. Pre-training each layer with an unsupervised learning method, starting with the first layer, can get a better training result with better regions in the parameter space. This is the common idea in both Restricted Boltzmann Machines (RBMs) and autoencoder training algorithms (two popular deep learning methods). More specifically, each layer is trained sequentially with an unsupervised learning algorithm [2], using as input the output of a previous layer. The higher level features extracted by unsupervised learning can be used as input or initialization to a supervised learning structure [1].

1.3 Scope and Objective

The main idea is to train the network with autoencoder and reinforcement learning together for an agent control task. The first part of the training,

Aim	Objective
Understanding the main principles of deep learning	Training a network with autoencoder and reinforcement learning
Application	Agent control task
Image classification	Agent should react to images with appropriate responses

Table 1: Aims and objectives of the project

that of the autoencoder, is meant to learn features of the visual inputs. The autoencoder neural structure is often used to build deep learning networks, and is reported to perform well in real-time control tasks [3]. Reinforcement learning using a modulated Hebbian rule [3] is a supervised learning. The features generated from the autoencoder are the input for the reinforcement learning. The control task is that the agent should react to images with correct responses. This shows the basic principle of the deep learning, which uses the features from the unsupervised learning (autoencoder) as the input of a supervised learning structure (reinforcement learning). Table 1 shows how the objective is related to the overall aim of the project.

1.4 Motivation

Human have rich experience in educating their descendants. Can this experience be applied to machine learning to improve the performance of learning algorithms? This issue is explored in this report. Playing could be considered as a learning process whose impact and utility to achieve certain survival goals are not often very clear. However, it is easy to hypothesis that it is a fundamental stage in learning. When a new concept is introduced to a child, the teacher often points out connections between the concept and what the child experienced from playing. This helps the child to understand the concept. The process of practicing is for the child to learn some concrete instances in the target field. We focus on this specific learning pattern and apply it on an agent control task, which lets an agent understand and react

to some visual concepts.

The problem considered in this study appears to require a two-stage computation, which is utilised by an agent to learn visual concepts in an control task (see Figure 1[6]). It is important to notice that reinforcement learning can occur only if higher level features are correctly extracted from low level visual inputs. Thus, learning appears to build up through stages. Accordingly, the first stage is to train the agent with unlabeled images of natural scenery . The agent learns features from the unlabeled images through unsupervised learning. These features are useful in the next stage, so they are stored to be transferred to the next stage.

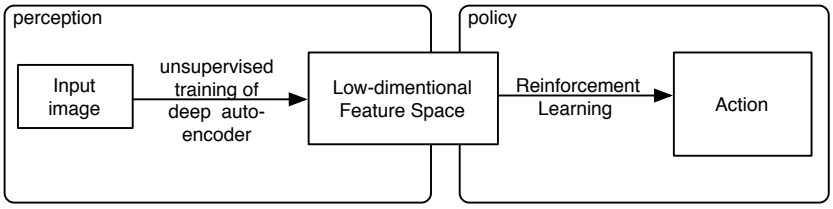

Figure 1: The two-stage framwork

The second stage is to use the reinforcement learning method to train the agent to perform tasks that result in a reward from the environment. The training in this stage is two-fold. The first fold is to train the agent with labeled images of representative directions to learn visual concepts of directions. The second fold is to refine the learning from the first fold so that the agent can identify the visual concepts from images of directional instances (understand the visual concepts in the target domain). Therefore, in the second fold, the agent is trained with a few of labeled images of directional instances.

The process of learning images of directional instances from the features learned from the images of natural scenery is the process of applying the knowledge in one domain to another domain, which is considered as transfer learning [4][5]. Recently, deep learning has shown good performance in transfer learning [1][2]. A current focus of research is in integrating deep learning and reinforcement learning in agent control tasks [3][6][7]. However, transfer

learning and vision concept learning [8] in an agent control task are not fully investigated.

1.5 Contribution and Related Work

A control task that has been addressed by means of the integration of an autoencoder and supervised learning is a navigation task in a multiple path environment [3]. The features learned in this control task are the data from the sensors (crumb sensors) in the path environment. The work presented in [3] is extended here by focusing in particular on the acquisition of features from images rather than low-dimensional crumb sensors in [3].

In some other works [6][7], integrating an autoencoder and a reinforcement learning for an agent control task has been used. The agent in those studies makes decisions on the control policies by evaluating its environment. A camera captures the images of the environment with the location of the agent. The images of the environment with the location of the agent serve as the guide for the agent to make control policies. A deep autoencoder is used to capture features from the images. Images are similar when the agent stays in the adjacent locations. Therefore, the agent determines its location by classifying images. The reinforcement learning is used to train the agent to make a good decision on different locations. However, in this project, the visual inputs are not particular instances of location, but rather general concepts describing directions. Thus, as opposed to [6][7], this study investigates the acquisition of higher level general concepts, rather than policies dictated by unique input data. The challenge in this project is not to determine the location of the agent, but to identify the visual concepts conveyed by images.

In order to investigate how general features and concepts can be learned, this study uses images of natural scenery that do not relate to the control task in which the agent engages later on in training. This is particular important because it demonstrates that the acquisition of features from various images and patterns can be of use also when performing on different images, which however share some common patterns. A framework called "self-taught

learning" [5] has used images of outdoor scenes to classify Caltch101 images (images of real life objects, such as an apple and a chair). The classification accuracy is promising. This gives the evidence of the possibility of using unlabeled nature images to learn features that can be of use later on for different tasks. After the first learning stage with natural images, handwritten direction lines are given to the agent as a focused input set to refine extraction of features that will be useful in the following control task. However, the task in this project is not the classification as in "self-taught learning"[5], but to identify the visual concepts.

The visual concept learning was proposed with Bayesian algorithms [8], but the way it learns visual concepts is different from the approach in this project, where a neural network is used for learning visual concepts for the control task.

The reinforcement learning in this project employs a modulated Hebbian rule [3]. However, a fresh meaning of the modulation factor in the rule is given in this report. The modulation factor in this report functions as a filter for irrelevant information when training with instances of visual concepts. With the modulation factor, the agent can tell if a patch of an image has the information of the visual concepts the image might convey.

2 Problem Definition

In a driving problem, an agent might respond to a visual scene when it is steering a vehicle. That is to say, the agent should decide how it steers the vehicle based on what it sees in front of it. Different visual scenes might convey the same visual concept. The agent might react with the same action to those different visual scenes. For example, the agent should stop when there is a stop sign on the road. The agent should also stop when there is a man in front of it posing a stop gesture. Although these scenes are different, they both convey the same concept, *stop*. Therefore, the concept conveyed by the visual scene that the agent captures is important. The appropriate control actions to be sent in order to drive correctly depends on the visual

concept. Thus, the control task for the agent is to react with the same action to every instance image of the same visual concept.

In this report, the visual concepts that an agent should learn are four directions, which are left, right, straightforward and horizontal(see Figure 2). An agent should react to the four visual concepts as turning left, turning right, going straightforward and keeping waiting respectively.

Handwritten images in Figure 3 are instances of the four visual concepts. The features in the instance that are relevant to driving are the orientations of the lines. Images No.1, No.5, No.9, No.13, No.17 and No.21 in Figure 3 are all instances of the same visual concept, the right direction. A successful agent should react as turning right to any given one of these images. The instances and their corresponding visual concepts are listed in Table 2.

The images of instances in the same visual concept usually do not appear similar to each other, but share the same characteristic. Learning visual concepts [8] is different from learning in facial recognition or classification of handwritten digits. In the visual concept learning, different objects might be regarded as in the same category. For example, images No.1 and No.5 in Figure 3 look different. The former has 7 lines whereas the latter has 1 line. However, they are instances of the same visual concept, the right direction. In contrast, images of different faces are often regarded as faces of different persons in the facial recognition, and images of different digits are regarded as different numbers in the classification of handwritten digits. However, a face is the face whether is observed frontally or sideways. Therefore, it can be concluded that visual concept learning is an ubiquitous principle in vision and control.

(a) Right (b) Left (c) Straightforward (d) Horizontal

Figure 2: 8×8 pixel images of representative directions.

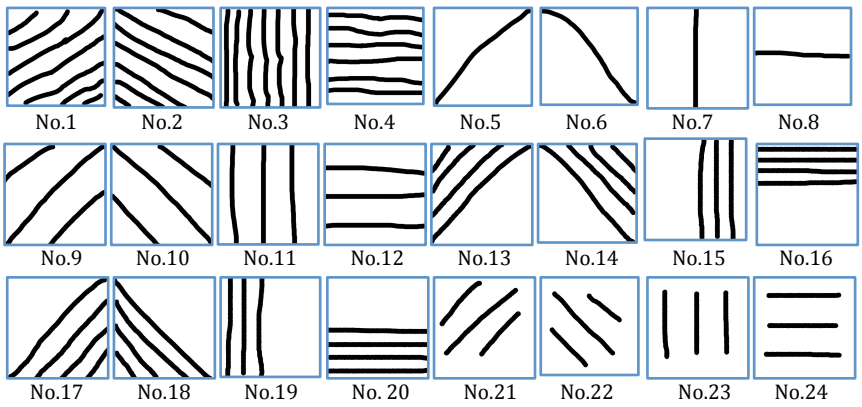

Figure 3: 512×512 pixel images of directional instances.

Instances	Visual Concept
No.1, No.5, No.9, No.13, No.17, No.21	Right direction
No.2, No.6, No.10, No.14, No.18, No.22	Straightforward direction
No.3, No.7, No.11, No.15, No.19, No.23	Left direction
No.4, No.8, No.12, No.16, No.20, No.24	Horizontal direction

Table 2: Instances and visual concepts

3 Methods

3.1 Learning Process

As introduced above, this study proposes a three-step learning process inspired by developmental learning [13]. The three steps are described following.

Traninig Step 1 This is an unsupervised learning process using an autoencoder algorithm. The agent learns features from unlabeled images of natural scenery. The images of natural scenery are images taken from outdoor environments[14]. These learning materials are similar to what a child sees when playing outdoor.

Traninig Step 2 This is a learning process that uses a reinforcement learning algorithm. The agent learns visual concepts from labeled images of representitive directions. The labeled image of a representitive direc-

tion is a small-size image that only consists of the key feature of one visual concept. Each visual concept has one labeled image of a representitive direction (Figure 2). The training data in this training step are similar to the learning materials for a child to learn from a teacher.

Traninig Step 3 This is also a learning process that uses a reinforcement learning algorithm. The agent learns visual concepts from labeled images of directional instances (Figure 3). The labeled image of a directional instance is an image that the agent is supposed to react to. Only a few of the labeled images are learned in this training step. The labeled images of directional instances are similar to the practicing material for a child to get familiar with the real problem.

Figure 4 summaries the three-step learning process. The input in the first step is the unlabeled images of natural scenery, the input in the second step is the labeled images of representative directions, and the input in the third step is the labeled images of directional instances. The features that the agent learned are shown in the images in the right hand in Figure 4. In the first step, vague features are learned. The stripes in the top right image are in random directions. In the second step, more precise concepts are formed. The stripes in the middle right image are in 4 directions. In the third step, the concepts the agent learned are refined. The directions in the bottom right are more clear than the one in the middle right image.

The learning results in Figure 4 can be stored as weights in a network structure. By training the weights with unsupervised (autoencoder) and supervised learning (reinforcement learning) in the network with the inputs in Figure 4, an agent can obtain the learned features (trained weights).

3.2 Network Structure

A three-layer neural network is built to learn visual concepts for the agent control task. Two groups of weights should be learned in this three-layer network, which are weights that connect the input layer to the hidden layer,

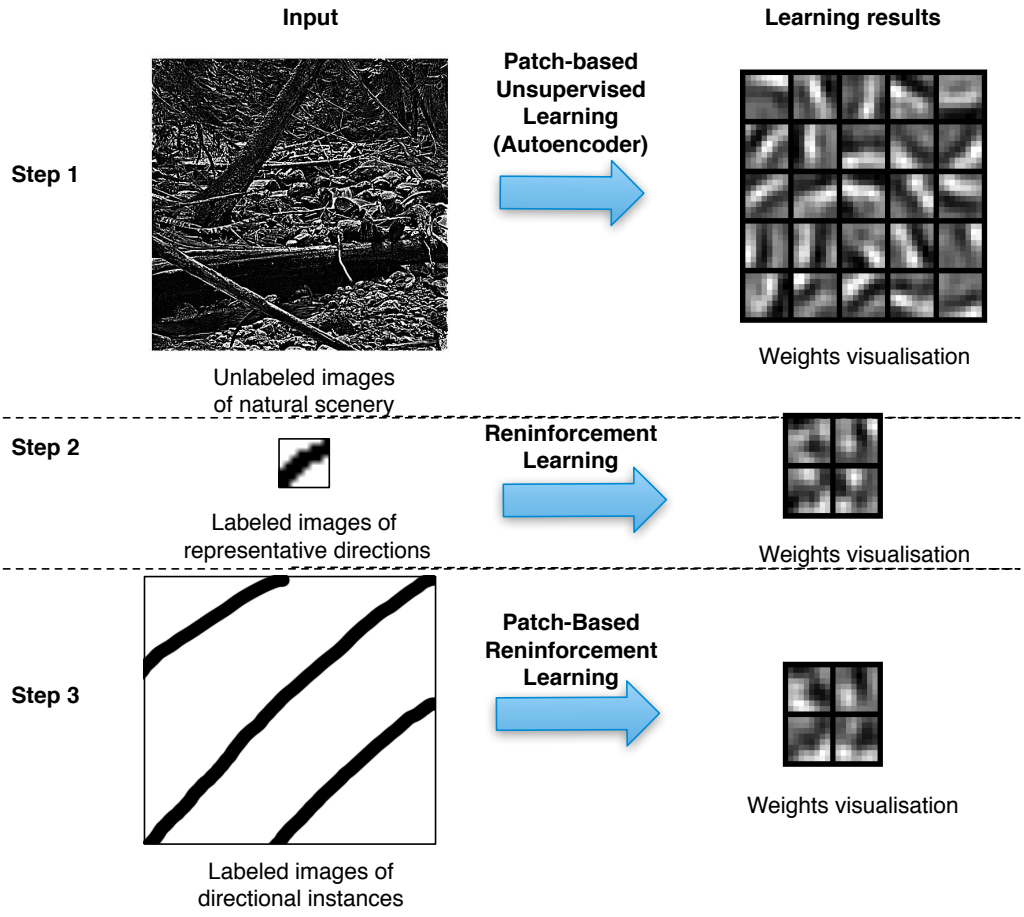

Figure 4: The three-step learning process.

and the weights that connect the hidden layer to the output layer. The training steps and weights update process in the network are shown in Figure 5. The autodencoder in the training step 1 is a structure that performs well

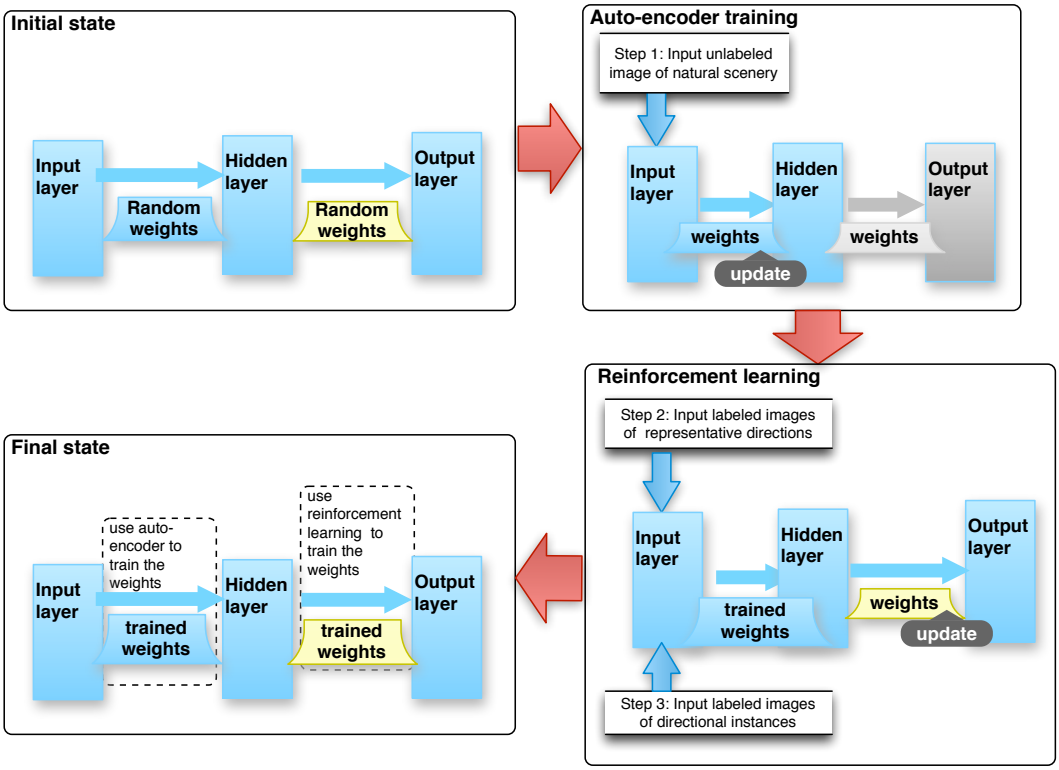

Figure 5: Network structure and training steps.

in reducing the dimensionality of data [9]. The idea of the autoencoder is to train a network with at least one hidden layer to reconstruct its input. Thus, the features in the hidden layer are what the agent learned from the input. The reinforcement learning in the training step 2 and training step 3 is used to adjust the connections between the features in the hidden layer and the output reaction. The idea of the reinforcement learning is to reward the connections when the output reaction responds correctly to the input, or to punish the connections when the output reaction responds incorrectly to the input.

The weights change as training process going. Figure 4 shows how the agent gets the visual concept clearer step by step. As shown in Figure 4, the

weights visualisation in step 1 are edge features that the agent learns from unlabeled images of natural scenery. These edge features are not like the ones in Figure 2, which clearly have direction patterns. The goal of the first training step is to let the agent get familiar with edges, even the edge features are vague. The weights visualisations in step 2 and step 3 have more clear direction patterns than the ones in step 1. It can be seen clearly in step 3 in Figure 4 that the top left corner of the weights visualisation looks like a left direction stripe, the top right corner looks like a straightforward stripe, the bottom left corner looks like a right direction stripe, and the bottom right corner looks like a horizontal direction stripe.

3.3 Agent Action

The action of the agent responding to an image depends on the ouput of the network. In the output layer, there are several output units. Each output unit is asscoiated with an action that is supposed to respond to a respective visual concept conveyed by the input image. The agent is designed to respond to the input with the action associated with the output unit whose output value is greater than the value of any other output unit.

3.4 Autoencoder Component

An autoencoder neural network is an unsupervised learning algorithm. It tries to learn a function that the output values of the function are almost the same as the input values.

In Figure 6, circles are used to represent the neurons (units) and bias units are labeled with "+1". Assume training examples $(x^{(i)}, y^{(i)})$ are aiven, the output result generated from the network is $h_{W,b}(x)$ with weight parameter W and bias parameter b. An autoencoder is to find $h_{W,b}(x) \approx x$ so that $\widehat{x}$ is similar to x. A compressed representation of the input is learned in the hidden layer. The number of the units in the hidden layer is usually smaller than the input layer. However, the units in the hidden layer contain key features of the input so that the output is similar to the input.

The sparse autoencoder method [10] used in this report has several components, which are feedforward, backpropagation, weight decay and sparsity penalty.

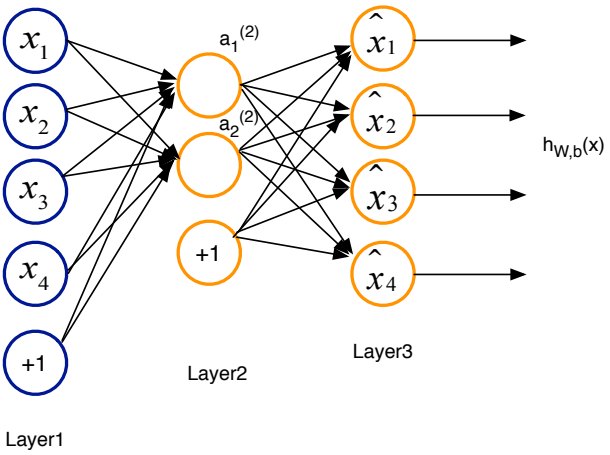

Figure 6: Illustration of a 3-layer Autoencoder.

3.4.1 Feedforward

The activation function $f(\cdot)$ chosen to implement in the network is sigmoid function:

$$f(z) = \frac{1}{1 + \exp(-z)}. \tag{1}$$

Activation $a_i^{(l)}$ stands for the output value of unit i in layer l. $a_i^{(1)} = x_i$ is used to denote ith input when $l = 1$. Let $z_i^{(l)}$ be the input of activation function in unit i in layer y, which means it is the the total weighted sum of inputs to unit i in layer l with bias term. The compact equations for feedforward process are

$$z^{(2)} = W^{(1)}x + b^{(1)} \tag{2}$$

$$a^{(2)} = f(z^{(2)}) \tag{3}$$

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)} \tag{4}$$

$$h_{W,b}(x) = a^{(3)} = f(z^{(3)}) \tag{5}$$

3.4.2 Backpropogation

Given a fixed training set $\{(x^{(1)}, y^{(1)}), ..., (x^{(m)}, y^{(m)})\}$ with m examples, let the error cost function for a single example be

$$J(W, b; x, y) = \frac{1}{2} \| h_{W,b}(x) - y \|^2 \tag{6}$$

and the error cost function for m examples is

$$J(W, b) = \frac{1}{m} \sum_{i=1}^{m} (\frac{1}{2} \| h_{W,b}(x) - y \|^2) \tag{7}$$

For one interation, parameters update as follows:

$$W_{ij}^{(i)} := W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b) \tag{8}$$

$$b_i^{(l)} := b_j^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W, b) \tag{9}$$

where,

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x^{(i)}, y^{(i)}) \tag{10}$$

$$\frac{\partial}{\partial bi^{(l)}} J(W, b) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial b_i^{(l)}} J(W, b; x^{(i)}, y^{(i)}) \tag{11}$$

The "error term" for the output unit in the output layer (layer 3) is

$$\begin{aligned} \delta^{(3)} &= \frac{\partial}{\partial z^{(3)}} \frac{1}{2} \| h_{W,b}(x) - y \|^2 \\ &= -(y - a^{(3)}) \cdot f'(z^{(3)}) = -(y - a^{(3)}) \cdot a^{(3)}(1 - a^{(3)}) \end{aligned} \tag{12}$$

and the "error term" for each node i in layer 2 is

$$\delta^{(2)} = \Big(\sum_{i=1}^{s_2+1} W_{ji}^{(2)} \delta^{(3)} \Big) f'(z^{(2)}) = \Big(\sum_{i=1}^{s_2+1} W_{ji}^{(2)} \delta^{(3)} \Big) a^{(2)}(1 - a^{(2)}), \tag{13}$$

where s_2 is the number of nodes in layer 2 (not counting the bias unit). Therefore, the desired partial derivatives:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) = a_j^{(l)} \delta_i^{(l+1)} \tag{14}$$

$$\frac{\partial}{\partial bi^{(l)}} J(W, b; x, y) = \delta_i^{(l+1)} \tag{15}$$

Let

$$\Delta W^{(l)} = \sum_{0}^{m} \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) \tag{16}$$

$$\Delta b^{(l)} = \sum_{0}^{m} \frac{\partial}{\partial b_{i}^{(l)}} J(W, b; x, y) \tag{17}$$

Then, the weight update function is

$$W^{(l)} = W^{(l)} - \alpha \frac{1}{m} \Delta W^{(l)} \tag{18}$$

$$b^{(l)} = b^{(l)} - \alpha \frac{1}{m} \Delta b^{(l)} \tag{19}$$

3.4.3 Weight Decay

The wight decay parameter λ controls the importance of the weight decay in the error cost function. It also prevent overfitting during training by decreasing the weight magnitude [11].The Equation (7) (error function)should be rewritten into

$$J(W, b) = \frac{1}{m} \sum_{i=1}^{m} (\frac{1}{2} \| \ h_{W,b}(x) - y \ \|^2) + \frac{\lambda}{2} \sum (W_{ji}^{(l)})^2 \tag{20}$$

and Equation (18) (weight update function) should be rewritten into

$$W^{(l)} = W^{(l)} - \alpha \Big[\frac{1}{m} \Delta W^{(l)} + \lambda W^{(l)} \Big] \tag{21}$$

3.4.4 Sparsity Penalty

By imposing a sparsity constraint on the hidden units, the autoencoder can discover interesting structure even if the number of the hidden units is large [10].

Let the average activation of hidden unit j

$$\widehat{\rho}_j = \frac{1}{m} \sum_{i=1} m[a_j^{(2)}(x^i)] \tag{22}$$

and let the average activation of each hidden neuron j to be close to zero,

$$\widehat{\rho}_j = \rho, \tag{23}$$

where ρ is a sparsity parameter (a small number, say 0.05).

The sparse penalty term is to achieve Equation (23). The sparse penalty term

$$KL(\rho \parallel \widehat{\rho}_j) = \rho log\frac{\rho}{\widehat{\rho}} + (1-\rho)log\frac{1-\rho}{1-\widehat{\rho}} \tag{24}$$

is based on Kullback-Leibler(KL) divergence [10]. $KL(\rho \parallel \widehat{\rho}_j) = 0$ if $\widehat{\rho}_j = \rho$.

The overall cost function is now

$$J_{sparse}(W,b) = J(W,b) + \beta \sum_{j=1}^{s_2} KL(\rho \parallel \widehat{\rho}_j), \tag{25}$$

where $J(W,b)$ defined in Equation (20), and β is the parameter that controls the weight of the sparsity term.

Specifically, the "error term" for the second layer (Equation (13)) is now become

$$\delta^{(2)} = \Big(\sum_{i=1}^{s_2+1} W_{ji}^{(2)} \delta^{(3)} \Big) f'(z^{(2)}) + \beta(-\frac{\rho}{\widehat{\rho}_i} + \frac{1-\rho}{1-\widehat{\rho}_i}). \tag{26}$$

3.5 Reinforcement Component

The connections between learned features and the outputs are trained with reinforcement learning. A modulated Hebbian rule [3] is used in this reinforcement learning, which is

$$\Delta w_i = \mu \eta x_i p, \tag{27}$$

where w_i is the weight of the connection between two neurons, x_i is the input data, p is the reward parameter, μ is the modulation associated with the training sample. When the output $x_i w_i$ has a correct respond to the correspond input, the reward should be a positive number; otherwise, the reward should be a negative number to punish the weight. If an input image contains little information of a visual concept, it is regarded as irrelevant input information. For example, a white background patch of an image conveys

nothing about the features, thus it is irrelevant input information for identifying visual concepts. The output $x_i w_i$ is small when a white background patch as input. A threshold value of the output $x_i w_i$ is set to decide whether the input is relevant or not (it is irrelevant when the output $x_i w_i$ is smaller than the threshold). In this way, this work takes the parameter μ as a filter for the irrelevant input information ($\mu = 1$ means the input is useful for a visual concept, whereas $\mu = 0$ means no learning for useless input).

4 Expriments

4.1 Programming

Programming language

In this project, the program is written in Matlab. Matlab is more efficient on matrix operations. The weights in the network structure can be stored in the matrix. Therefore, Matlab is an appropriate language for this project. Besides, there are tutorials about autoencoders available on the Internet. Matlab codes about gradient checking (a method checking for the correctness of the back propagation) and L-BFGS (an optimization method for training) are available in the tutorial [14], which is helpful for writing a more reliable code of the autoencoder. Another programming language option is Python, which is not used in this project. There is a tutorial about deep learning that uses Python as the programming language [15].

Code structure

The flow chart of the code is shown in Figure 7.

Data

All labeled images are drawn in the software Pixelmator with fingers moving on the track pad. The unlabeled images are from a tutorial [14].

Normalization

The normalization for unlabeled images of natural scenery and labeled images are different. The unlabeled images of natural scenery are normalized

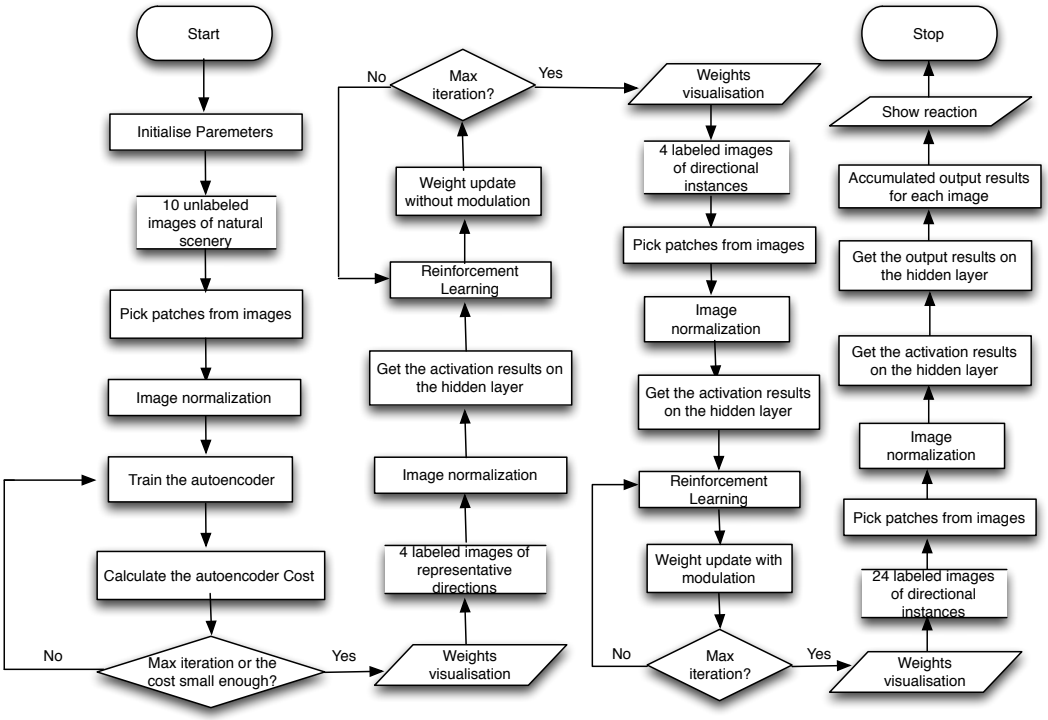

Figure 7: Flow chart of the complete learning algorithm developed in this project and code in Matlab.

by a feature standardization method [16]. Removing the mean-value per example in the feature standardization method can help feature learning [16] from images with rich information (such as the unlabeled images of natural scenery). The labeled images are normalized by a simple rescaling method [16]. Patches from the labeled images are black and white, simple rescaling method is more suitbale.

Limitations

The code can only deal with back and white labeled images for now.

4.2 Parameter Setting

For the specific problem here, the three-layer neural network has 64 units in the input layer, 25 units in the hidden layer, and 4 units in the output layer. When the size of an input image is bigger than 8×8, a large quantity of 8×8 patches are randomly picked from the image as input (Figure 8 is an example). This is called patch-based training, which learns features from local patches extracted at random positions of the inputs [12]. The number of units in the input layer and the hidden layer are references to a tutorial[10]. The 4 output units represent the 4 actions. The agent should react with the action associated with the biggest output unit .

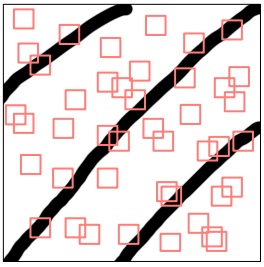

Figure 8: Random 8×8 patches on a 512×512 labeled image of an instance of a visual concept.

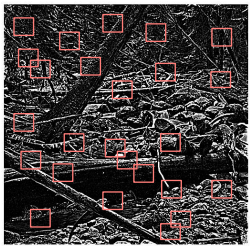

Figure 9: Random 8×8 patches on a 512×512 unlabeled image of natural scenery.

The following is the training process.

Traninig Step 1 An agent observes 10 512×512 unlabeled images of natural scenery (10000 8×8 patches are randomly picked from the images) and obtains edge features through unsupervised learning (autoencoder). Figure 9 shows how patches are picked from a natural image.

Traninig Step 2 The agent observes 4 8×8 labeled images of representative directions (see Figure 2) and learns how to react through reinforcement learning (400 times training). Here, the modulation parameter $\mu = 1$ in the reinforcement learning .

Traninig Step 3 The agent observes 4 512×512 labeled images of directional instances (Instances No.9-No.12 in Figure 3) (240000 patches are randomly picked) and learns how to react through the reinforcement learning. When the biggest output unit in the output layer has a value smaller than the threshold (it is set as 0.15 because white pixels are normalized to 0.1 in this project, and 0.15 is a proper value for filtering white patches through experiments), it means the input might be a white background. Background information is useless for identifying visual concepts, so the modulation parameter $\mu = 0$ in the reinforcement learning. If it is not the case, set $\mu = 1$.

Testing method

24 512×512 labeled images of directional instances in Figure 3 (1000 patches are randomly picked in each image), where 20 of them haven't been seen by the agent before, are use to test how well the agent learned. For each image, the accumulated value with the 1000 input patches in the 4 output units are calculated. The unit with the biggest accumulated value is the activation unit. The agent should response to the input image with the action associated with the activation unit.

Run time

The total run time (training and testing) is 170.05 seconds. It took 30.36 seconds to train the autoencoder in the training step 1, 0.24 seconds in the

reinforcement learning in the training step 2, 32.06 seconds in the patch-based reinforcement learning in the training step 3. This seems a fast training. The training method seems to have a potential ability to be expanded to more complex problems. It can be used in bigger networks with more images in the future.

4.3 Exploring Results in Autoencoder

Figure 10 shows the activation results on the hidden layer with the 4 labeled images of representative directions (images in Figure 2) as inputs. The bigger the output value in a hidden unit, the more similar the input image is to the hidden unit. Unit 1 in Figure 10 has four output values. The first output value shows that the feature the unit 1 stands for is merely related with the first input (the label image of the right representative direction, see Figure 2(a)).

As shown in Figure 10, the first input image has a large value in units 10, 18, 22, 23 and 25 (these units are circled in orange in Figure 11). As in Figure 11, features that units 10, 22, 23 and 25 stand for are in a right direction, which are instances of the same visual concept that the input image conveys. The feature the unit 18 stands for does not show an obvious direction pattern. However, the overall activation result from training autoencoder reveals a strong relation between the input images and their features, which shows the features generated in the autoencoder can help the agent to understand visual concepts correctly.

4.4 Exploring in Reinforcement Learning

Figure 11 shows how the reinforcement learning (in training step 2) is related to the results of the autoencoder. The input image in the reinforcement learning (in training step 2) is the combination of the features generated in the autoencoder. Features in the orange circle in Figure 11 have the highest activation values for the labeled image of representative right direction . Features from the autoencoder can be considered as experience from observing

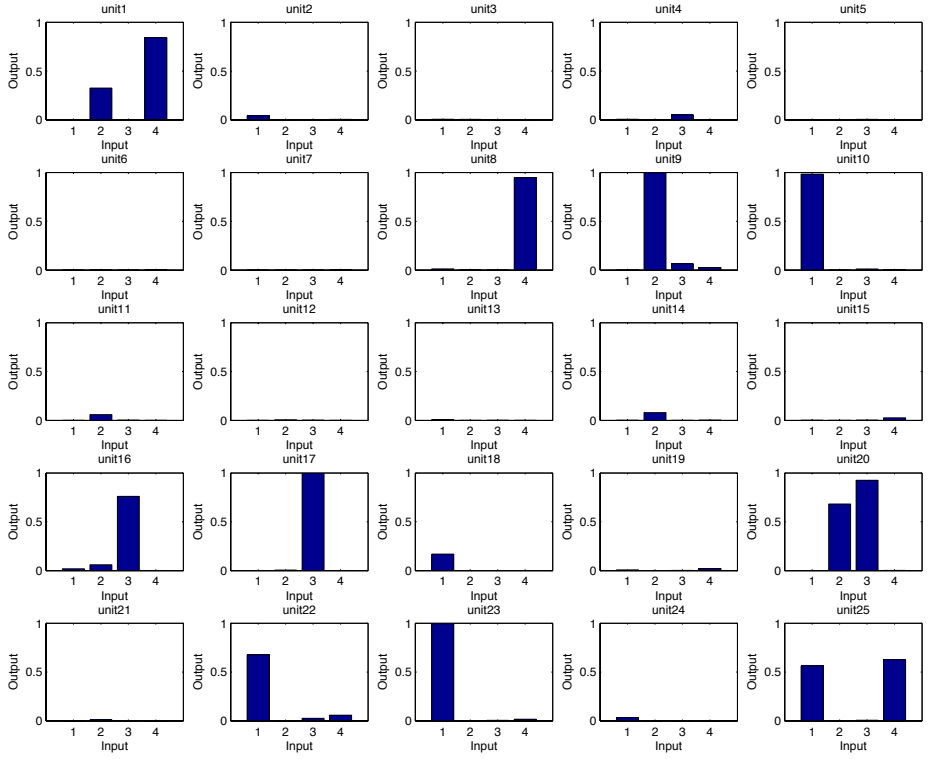

Figure 10: Activation on the hidden layer with 4 labeled images of representative directions as inputs. The output is the activation value on the hidden layer.

the environment. The input images are the label images of the representative directions. Figure 11 shows that the input image of the right representative direction can be calculated from the combination of the features. This is how the visual concepts connect to the natural images.

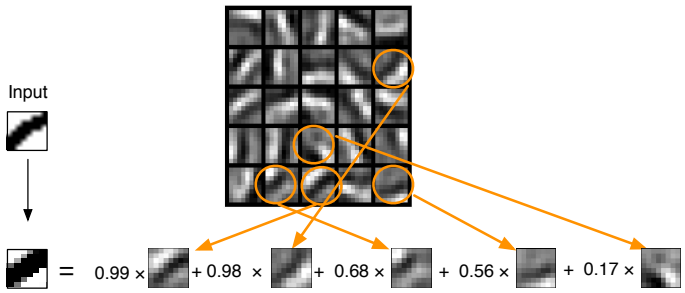

Figure 11: Features computed for a labeled image of representative right direction (left) by representing the labeled image as a sparse weighted combination of features (right). These features are generated in the autoencoder. Features from top to bottom, left to right correspond to the hidden units 1,2,3,...,25 respectively. The coefficient numbers are the activation values in the corresponding hidden units.

4.5 Exploring Effectiveness in Patch-based Reinforcement Learning

In the patch-based reinforcement learning, labeled images of directional instances are used as training data. The weights for the output layer are adjusted during the patch-based reinforcement learning, see Figure 12-15. As shown in Figure 12, weights connecting hidden units 4, 9 and 19 to the output unit, which represents the left direction, become bigger. It can be seen in Figure 11 that the units 4, 9 and 19 are all in a left direction. These adjustments strengthen the connections between the features and the correct visual concepts. On the other hand, the weights connecting units 1, 10 and 12 with the output unit decrease in the negative way. In Figure 11, the unit 1 is slightly in a horizontal direction, the unit 10 is slightly in a straightforward direction, and the unit 12 is in a right direction. Therefore, the adjustments on these units enlarge the irrelevant distance between the incorrect features

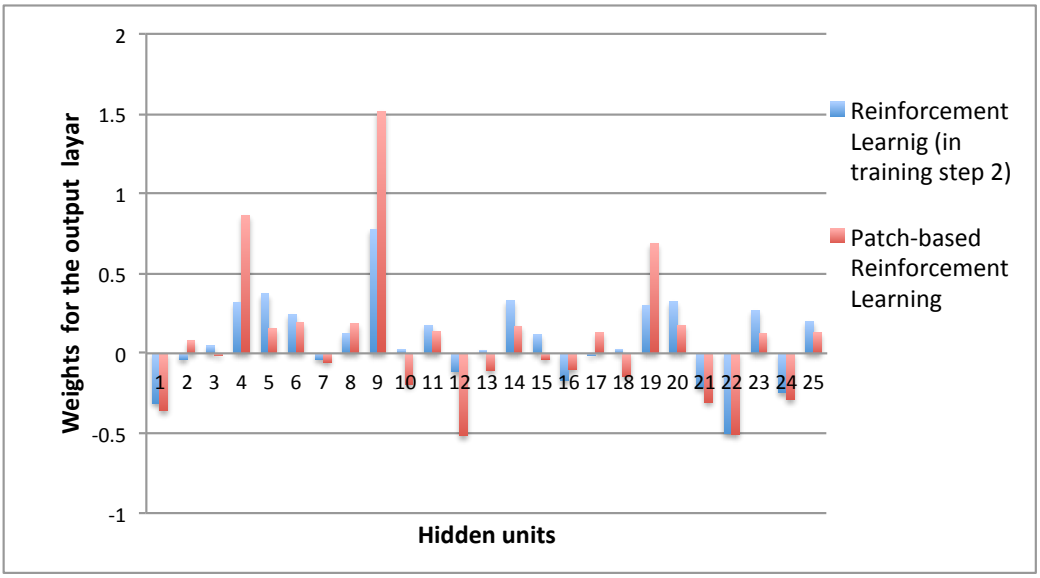

Figure 12: Weights connect to the left concept output unit. Weights changed from reinforcement learning (in training Step 2) to patch-based reinforcement learning.

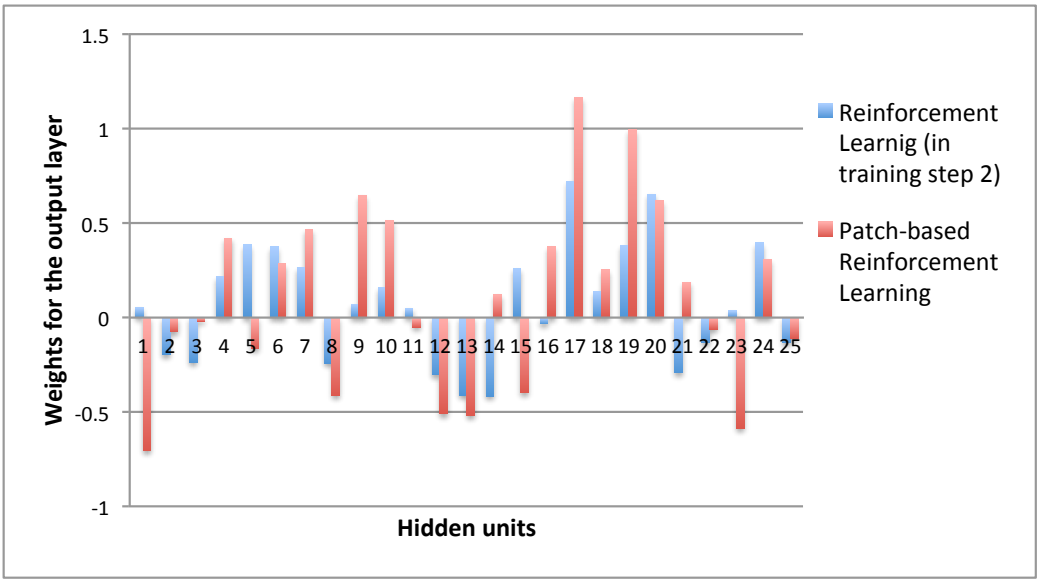

Figure 13: Weights connect to the straightforward concept output unit. Weights changed from reinforcement learning (in training Step 2) to patch-based reinforcement learning

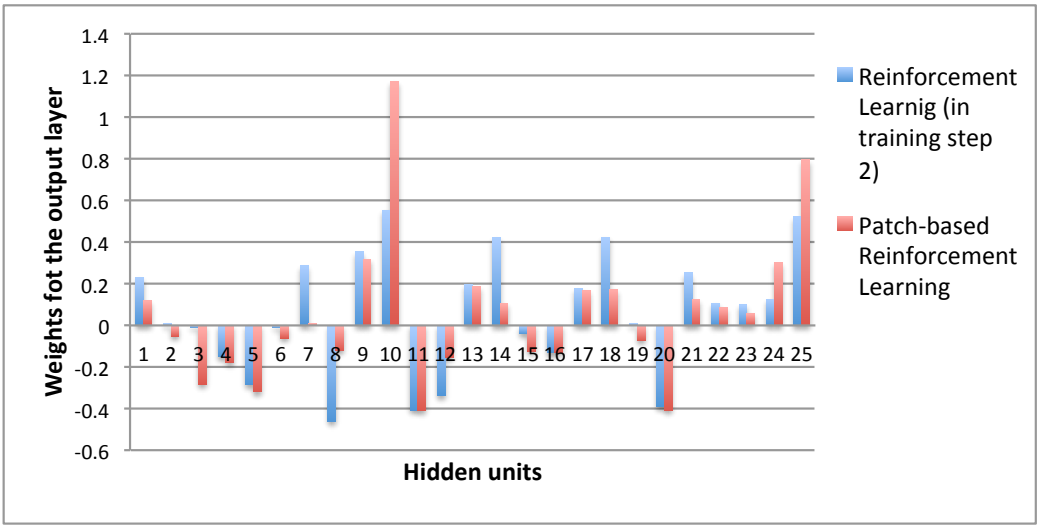

Figure 14: Weights connect to the right concept output unit. Weights changed from reinforcement learning (in training Step 2) to patch-based reinforcement learning

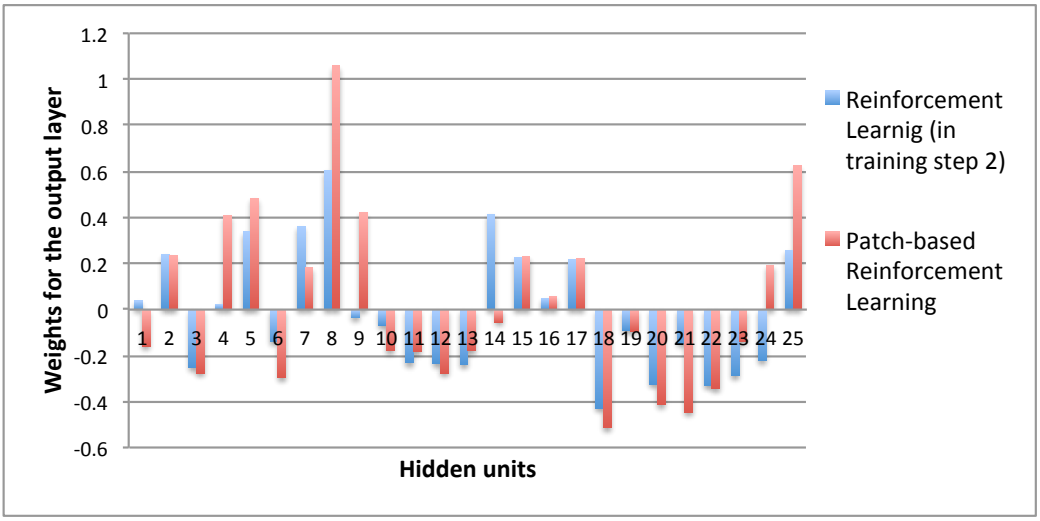

Figure 15: Weights connect to the horizontal concept output unit. Weights changed from reinforcement learning (in training Step 2) to patch-based reinforcement learning

and the output unit.

5 Results and Disscussion

All 24 labeled images of directional instances are understood by the agent correctly after training the network with 10 unlabeled images of natural scenery, 4 labeled images of representative directions and 4 out of 24 labeled images of directional instances. In other words, the accuracy rate is 100%.

Using 4 labeled images of directional instances in the autoencoder in the training step 1 instead of unlabeled images of natural scenery (replace the input in the autoencoder with unlabeled images of natural scenery), the accuracy rate is 29.2%. Without training 4 labeled images of representative directions in the reinforcement learning in the training step 2 (skip the reinforcement learning in the training step 2), the accuracy rate is 58.3%. Without training 4 labeled images of directional instances in the patch-based reinforcement learning in the training step 3 (skip the patch-based reinforcement learning in the training step 3), the accuracy rate is 33.3%. As we can see from the results (Table 3), the results get worse when we skip any one of the training steps in the learning process.

The following subsections will discuss the results further.

Training approach	Accuracy
Take all the 3 training steps	100%
Replace the input in autoencoder with labeled images of directional instances.	29.2%
Skip reinforcement learning	58.3%
Skip patch-based reinforcement learning	33.3%

Table 3: Test results from different training approches.

5.1 Why Unlabeled Images of Natural Scenery?

When unlabeled images of natural scenery are replaced with labeled images of directional instances in the autoencoder, the result gets worse. This is

because the labeled images of directional instances have less edge features and more background patches. When the agent scans some amount of random patches on the labeled images of directional instances, it will get confused with the large amount of background patches and have difficulty in detecting major features.

Figure 16 shows the features in the hidden layer when 4 labeled images of directional instances are used as training data in the autoencoder. It is obvious that features in Figure 16 are so vague to be identified. Edge features in Figure11 are far more clear than the features in Figure 16.

This phenomena reveals the utility of having a rich input images to learn general features in a robust and reliable way.

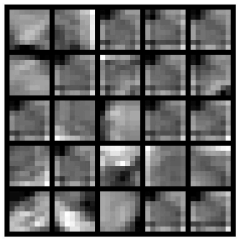

Figure 16: Weights visualisation in the autoencoder using labeled images of directional instances .

5.2 Why Labeled Images of Representative Directions?

As it been discussed above, the labeled images of directional instances have more background information which will interfere with the agent while it is identifying the major features in these images. In the reinforcement learning stage, this factor should also be taken into account.

An effective way of initializing the weights is to pre-train the network to make the weights close to a good solution [9]. This is the utility of training labeled images of representative directions in the reinforcement learning stage.

When using only the labeled images of directional instances, the agent will

get a large amount of patches with useless information (white background). Furthermore, patches in the labeled images of directional instances often contain features that are not strictly in the representative direction. This disturbs the agent to adjust the weights in the neural network in a promising way. Using labeled images of representative directions guarantees that the input images have obvious features of the visual concepts. The reward process in the reinforcement learning is more accurate.

5.3 Why Labeled Images of Directional Instances?

The labeled images of directional instances are handwritten, so that the stripes in the images are not strictly straight. By picking patches in the labeled images of directional instances, different styles of small patches are taken into consideration for identifying the visual concepts. In this way, the weights for the output layer are adjusted to handwritten stripes instead of straight stripes in the reinforcement learning in the training step 2.

The adjustment in the patch-based reinforcement learning in the training step 3 is slightly, see Figure 12-15. However the impact on the results is big, see Figure 17 and Figure 18.

For each image, the average value of the accumulated values in each output unit is generated. Four average values are generated in the four output units. These four values show how related the image is with the four visual concepts the output units represent for. The bigger the value is, the higher possibility that the image conveys the visual concept the output unit stands for. The output unit with the biggest value is the activation unit for the input image. As shown in Figure 17, "turning right" is the reaction for the labeled image of the directional instance No.1, and "going straightforward" is for the labeled image of the directional instance image No.3.

Figure 18 shows the results in the reinforcement learning in the training step 2. The biggest value for each input is very different from the one in the patch-based reinforcement learning in the training step 3. The reaction for the first input is "turning right" in the patch-based reinforcement learning

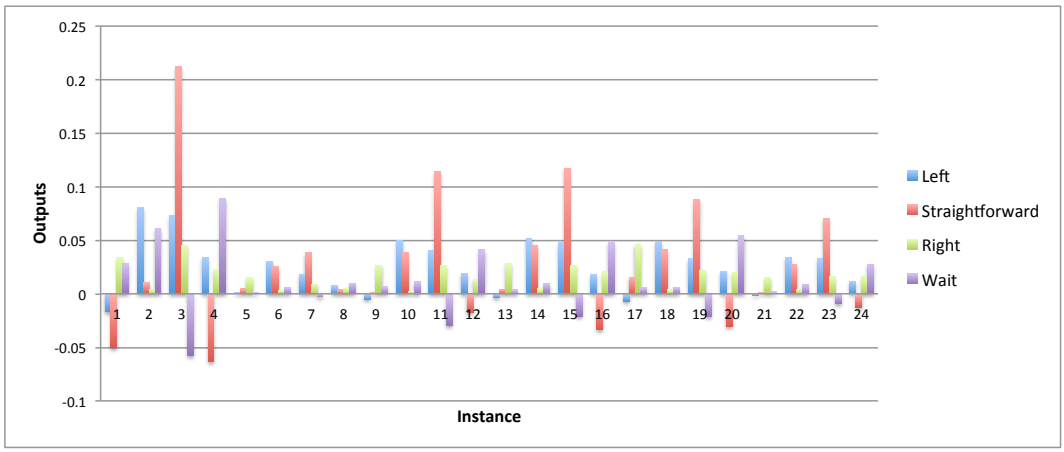

Figure 17: Outputs for 24 instances in the patch-based reinforcement learning in the training step 3.

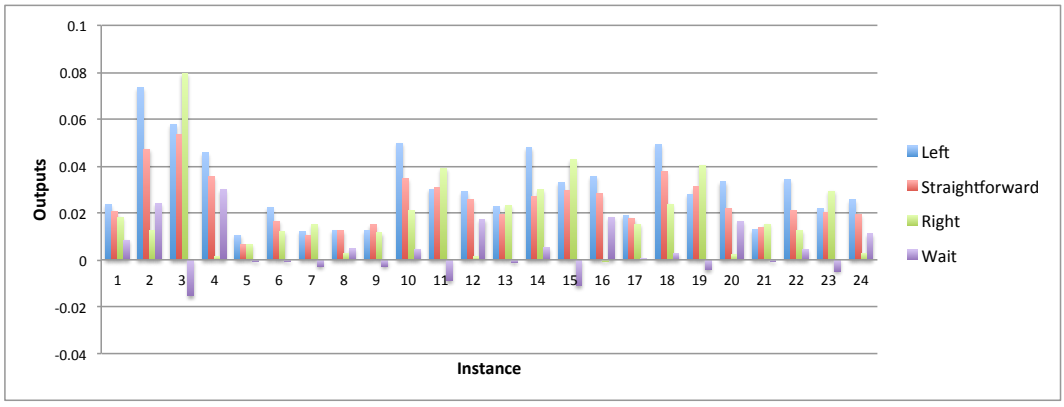

Figure 18: Outputs for 24 instances in the reinforcement learning in the training step 2.

in the training step 3, whereas it is “turning left ” in the reinforcement learning in the training step 2. The reaction for the third input is “going straightforward” in the patch-based reinforcement learning in the training step 3, whereas it is “ turning right” in the training step 2. Moreover, the output values in the reinforcement learning in the training step 2 are smaller than the ones in the training step 3, which shows that the results in the patch-based reinforcement learning in the training step 3 are more reliable.

6 Conclusion

In this report, a novel approach of learning visual concepts for a control task is proposed. This approach takes a common learning experience in the real world into consideration and performs well in the control task. A two-stage framework with three training steps is implemented. It is successful on a three-layer network. The necessity of three training steps and the input data in those training steps are discussed. The experiments' results show that an agent can have the same reaction to different images that convey the same visual concept. It is promising that more images that convey visual concepts can be identified with this method, which is helpful in a robot control with visual instructions. The next step will be to apply this approach to a real world system. In the real word system, the situation is more complicated. More complex visual concepts and instances need to be identified by the robot. The autoencoder will have deep layers when abstacting features from more complex images of visual instances. The chanllenge will be the noise and uncertainties in the real enviornment.

References

[1] Bengio, Yoshua, Aaron Courville, and Pascal Vincent. "Representation learning: A review and new perspectives." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 35.8 (2013): 1798-1828.

[2] Bengio, Yoshua. "Learning deep architectures for AI." *Foundations and trends® in Machine Learning* 2.1 (2009): 1-127.

[3] Pugh, Justin K., Andrea Soltoggio, and Kenneth O. Stanley. "Real-time Hebbian Learning from Autoencoder Features for Control Tasks." *ALIFE 14: The Fourteenth Conference on the Synthesis and Simulation of Living Systems.* Vol. 14.

[4] Pan, Sinno Jialin, and Qiang Yang. "A survey on transfer learning." *Knowledge and Data Engineering, IEEE Transactions on* 22.10 (2010): 1345-1359.

[5] Raina, Rajat, et al. "Self-taught learning: transfer learning from unlabeled data." *Proceedings of the 24th international conference on Machine learning.* ACM, 2007.

[6] Lange, Sascha, and Martin Riedmiller. "Deep auto-encoder neural networks in reinforcement learning." *IJCNN.* 2010.

[7] Lange, Sascha, Martin Riedmiller, and A. Voigtlander. "Autonomous reinforcement learning on raw visual input data in a real world application." *Neural Networks (IJCNN), The 2012 International Joint Conference on.* IEEE, 2012.

[8] Jia, Yangqing, et al. "Visual concept learning: Combining machine vision and bayesian generalization on concept hierarchies." *Advances in Neural Information Processing Systems.* 2013.

[9] Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." *Science* 313.5786 (2006): 504-507.

[10] Ng, Andrew. "Sparse autoencoder." *CS294A Lecture notes* 72 (2011).

[11] Hinton, Geoffrey. "A practical guide to training restricted Boltzmann machines." *Momentum* 9.1 (2010): 926.

[12] Coates, Adam, and Andrew Y. Ng. "The importance of encoding versus training with sparse coding and vector quantization." *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011.

[13] Elman, Jeffrey L. "Learning and development in neural networks: The importance of starting small." *Cognition* 48.1 (1993): 71-99.

[14] Ng, Andrew. "Exercise:Sparse Autoencoder". Available on `http://ufldl.stanford.edu/wiki/index.php/Exercise:Sparse_Autoencoder`.

[15] LISA lab, University of Montreal. "Deep Learning Tutorial Release 0.1 ". Available on `http://deeplearning.net/tutorial/deeplearning.pdf`.

[16] Ng, Andrew."Data Preprocessing". Available on `http://ufldl.stanford.edu/wiki/index.php/Data_Preprocessing#Data_Normalization`