Teaching Machine Learning to Design Students

Bram van der Vlist, Rick Westelaken, Christoph Bartneck, Hu Jun, Rene Ahn, Emilia Barakova, Frank Delbressine, Loe Feijs

Department of Industrial Design
Eindhoven University of Technology
Den Dolech 2, 5600MB Eindhoven
The Netherlands

[h.f.m.v.d.westelaken; b.j.j.v.d.vlist]@student.tue.nl; [c.bartneck; j.hu; r.m.c.ahn; e.i.barakova; f.l.m.delbressine; l.m.g.feijs]@tue.nl

Abstract

Machine learning is a key technology to design and create intelligent systems, products, and related services. Like many other design departments, we are faced with the challenge to teach machine learning to design students, who often do not have an inherent affinity towards technology. We successfully used the Embodied Intelligence method to teach machine learning to our students. By embodying the learning system into the Lego Mindstorm NXT platform we provide the student with a tangible tool to understand and interact with a learning system. The resulting behavior of the tangible machines in combination with the positive associations with the Lego system motivated all the students. The students with less technology affinity successfully completed the course, while the students with more technology affinity excelled towards solving advanced problems. We believe that our experiences may inform and guide other teachers that intend to teach machine learning, or other computer science related topics, to design students.

Introduction

The Department of Industrial Design at the Eindhoven University of Technology prepares students for a new type of engineering discipline: design and creation of intelligent systems, products, and related services. These systems, products and services require adapted to the user and thereby provide a new experience. In the framework of our Masters program, we offer a course that familiarizes students with a number of powerful conceptual and intellectual tools to understand and create adaptive behavior at a system level.

System level thinking has had and still has an enormous impact upon the development of technology. When working at a system level one does not study individual component behavior, such as Ohm's law for an electrical component; instead one addresses bigger questions such as the stability of the feedback loops, information throughput, or learning capacity. The learning objective includes classical control, reinforcement learning and adaptive control, pattern recognition. The context of Lego is chosen because it is already an example of a system. The project’s creative goal is to make a leap forward, extending the scope of the existing system such that adaptive behavior becomes the central theme.

Like many other design departments, we are facing the challenge of teaching the mathematical foundation of machine learning to students that are neither mathematicians nor computer scientists. Most of the students in our department do not

have an inherent affinity towards technology. They do not build up in depth knowledge of programming or math.

One of the difficulties in teaching machine-learning is that its theory is abstract. The process and the results of the machine learning are only available inside a computer program. Design students are used to create and work with artifacts in the real world, not with mathematical formulas. This abstraction level inhibits their understanding and makes it difficult for them not only to reproduce relevant knowledge, but also to apply and extend it.

We therefore created a new teaching method to better support students in their learning of machine learning. Our new method involves the usage of embodiment intelligence; transferring the abstract theory into a more hands-on experience. We will elaborate on the structure of the course, the materials used, and two concrete case studies. Our method is not limited to machine learning, but can be used to teach many other aspect of computer science to design students. We believe that our insights may inspire and guide other teachers to create better courses for their design students.

Structure of the course

The course's first two weeks are theory oriented. A week during this phase typical consists of two days of theory at the start, followed by three days of practice with intermediate moment of contact between students and teachers to discuss their progress and to answer specific questions. In these two weeks the students work on very specific methods and principles. During the third and forth week the students are invited to demonstrate their understanding of the theory through something that they create. The teachers encourage depth, through additional theory, tools and methods.

We will now provide a more in depth view on the content of the course, but we would like to emphasize that the method may also be applied to teach different aspects of computer science. In our specific course, the goal is to teach the principles of reinforcement learning and supervised learning to design students.

Embodied Intelligence

We selected Q-learning and Neural Networks as basic examples of reinforcement learning and supervised learning. We embedded this form of intelligence into a real body: the Lego Mindstorms NXT. Lego Mindstorms is an excellent prototyping platform (Bartneck & Jun, 2004) for creating embodied intelligence. The platform features a NXT brick that includes a microprocessor capable of running a Java virtual machine. It comes packaged with several plug-and-play sensors and actuators and is, by definition, compatible with the Lego brick system. Prototypes can be built with click-and-connect ease, which allows students to focus on the implementation of the software.

Traditionally, machine learning (Bishop, 2006) is demonstrated through a computer program that does not only have to perform the learning, but which also has to simulate the environment on which the input for the learning model is based. By using an embodiment, such as the NXT, the sensory input does not longer need to be simulated. The learning program receives its input directly through the attached sensors that react to the stimuli that are already available in the real world (Nehmzow, 2003). The learning system could, for example, try to learn from the light sensor that is mounted on the bottom of a robotic car. The goal of such a learning program would be to learn how to follow a black line on the ground. The real world can offer a

richness that would be difficult to simulate. In addition, the embodiment allowed the students to easily explore the influence of the various variables. This simplified and enriched the process of understanding the meaning of variables in an algorithm, seeing the results in behavior change of the embodiment.

Participants

The participants of our course are all industrial design master students, which can be classified into two types. The first group consists of students that have a certain affinity with technology. These students like to explore, what are for them, new technological principles. They have a good understanding about a wide range of technologies and their applications. They also have considerable programming skills, with JAVA as solid basis. This group of students is usually the smaller of the two groups and teaching them machine learning is easier. They might even be satisfied with the traditional non-embodied method, but using the Lego NXT platform considerably increases their motivation.

The second group of students can be described as students that do not have an inherent affinity with technology. They have a limited understanding of technological principles and master programming only up to a basic level. Teaching these students machine learning is the true challenge.

It still needs to be acknowledged that students of either type are not mathematicians or computer scientists. These students are used to the creative creation of artifacts and not to formulas and logarithms. The teaching method needs to adapt to these characteristics.

Material

For an embodied intelligence course software and equipment is necessary. While the software is available for free, the hardware does require a certain budget. The basic Lego Mindstorms Education NXT set is currently available for 285 Euro. Our practical experience shows that one set can be shared by a maximum of two students. We will now discuss the required hardware and software in more detail.

Hardware

The NXT brick is part of the Lego Mindstorms set. The NXT is an embedded system with a plastic casing compatible with the Lego brick system. This way it can easily be integrated into a Lego construction that may also contain the sensors and actuators (Gasperi, Hurbain, & Hurbain, 2007). Lego saves a lot of time in constructing mechanical components compared to other methods. An educational version is available that includes the useful rechargeable battery, a power supply and a storage box. The NXT specifications are:

- Atmel 32-bit ARM main processor (256 Kb flash, 64 Kb RAM, 48 MHz)
- Atmel 8-bit AVR Co-processor (4 Kb flash, 512 Byte RAM, 8 MHz)
- Bluetooth wireless communication (CSR BlueCoreTM 4 v2.0 +EDR System)
- USB 2.0 communication (Full speed port 12 Mbit/s)
- 4 input ports: 6-wire interface supporting both digital and analog interface
- 1 high speed port, IEC 61158 Type 4/EN 50170 compliant

- 3 output ports: 6-wire interface supporting input from encoders
- Display: 100 x 64 pixel LCD black & white graphical display
- Loudspeaker: Sound output channel with 8-bit resolution (Supporting a sample rate of 2-16 KHz)
- 4 button user-interface
- Power source: 6 AA batteries or rechargeable Lithium-Ion battery.

Lego has developed a number of sensors and actuators as part of the Lego Mindstorms set. All these sensors are compatible with the Lego brick system. The basic Lego NXT Education set contains the following sensors and actuators:

- Touch sensor – detects when it is being pressed by something and when it is released again.
- Sound sensor – detects both decibels [dB] and adjusted decibel [dBA].
- Light sensor – reads the light intensity in a room and measure the light intensity of colored surfaces.
- Ultrasonic sensor – measure distances from 0 to 255 centimeters with a precision of +/- 3 cm.
- Servo motor with build in rotation sensor.

As result of the success of the Lego Mindstorms, other companies developed additional sensors and actuators. Some of these companies, such as HiTechnic Products and Mindsensors.com, provide sensors for the NXT platform such as IR Link Sensor, Gyro Sensor, IR Seeker Sensor, Compass Sensor, Color Sensor, Acceleration / Tilt Sensor, Magnetic Compass, and Pneumatic Pressure Sensor. In addition to the Lego NXT set, a standard computer is needed to write the programs. The programs are then uploaded to the NXT using either USB or Bluetooth.

Software

Three software components are necessary for this course. All of them are available for free and they replace the original Lego software. Lego's own software development tool is targeted to children and hence does not offer the flexibility and extendibility required for a university course. An extensive tutorial on how to install the components is available at: http://www.bartneck.de/work/education/masterClassLego/javaInstallNXT/. We will now describe the components in detail.

Java is a platform independent, object-oriented programming language (http://www.sun.com/java/). The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to bytecode, which can run on any Java virtual machine (JVM) regardless of computer architecture. It is a popular language for embedded systems, such as micro controllers and mobile phones and also the Lego NXT is able to execute Java programs.

It is advisable to use an integrated development environment (IDE) to write Java programs. Eclipse is the powerful and widely used IDE that offers excellent support for Java and the Lego NXT. Eclipse itself is written in Java and its installation is particularly easy.

To enable the Lego NXT to execute Java programs, its original firmware needs to be replaced with the open source leJOS firmware (Solorzano, 2002). The old firmware can be reinstalled at any point. Conveniently, leJOS includes a Java Virtual Machine so that no further software installations on the NXT are necessary to execute Java programs. The leJOS Java library is an extension to the standard Java and enables Java programs to use the platform specific features of the NXT, such as sensors and actuators.

The Java Object Oriented Neural Engine (Joone) is an application that allows users to build, test and train neural networks (http://www.joone.org). It features a convenient graphical user interface. Neural networks trained with Joone can be exported and called from any external Java program. It can therefore easily be integrated into more general Java programs, such as Java programs for the Lego NXT.

Case Study 1: Reinforcement Learning with the Crawler

During one week, the students mounted the NXT brick on wheels and gave it an arm with two Lego NXT electronic motors, creating the crawler (see Figure 1). This crawler has wheels (not driven) to freely move forward and backward. In order to move itself, the crawler can only use its arm, which has two joints under motor control. The Crawler has sensors to measure the position of the joints of the arm and also one distance sensor which “sees” the distance from a wall or another reference object. The NXT brick was programmed in Java to execute the reinforcement learning algorithm (Q-learning). It is positive rewarded if it moves forward and negative rewarded if it moves backwards. It explores its possibilities and learns how it should move to accumulate a maximal reward. The Crawler starts from seemingly random movements, but after a few minutes really finding a kind of rhythm how to move the arm and efficiently move forward.

[figure]

Figure 1: The Crawler

We will now discuss the Q-learning theory in more detail to enable the reader to form a better judgment of the difficulty that the students were able to overcome during one week by our teaching method. Q-learning is a common and well known reinforcement learning algorithm (Sutton & Barto, 1998). This method of machine learning is based on the principle of rewarding (Watkins, 1989). When a machine performs an action in a certain state it can get a positive reward, negative reward (punishment) or no reward. This depends on the design and goal of the machine. This way, the machine builds up an action-value table. The Q-learning logarithm works by using an action-value function that gives the expected value of taking a given action in a given state. This happens by following a certain policy. This policy describes if the machine should exploit its knowledge and chose the actions that lead to the biggest reward or that it should explore new actions in certain states to discover better ways to retrieve even more rewards later on.

The strength of Q-learning is that it will adapt to its environment without knowing it and without that it is programmed. Q-learning, as well as other reinforcement learning principles, works because it tries to optimize a given reward. Q-learning requires a finite set of environment states, a fixed set of actions, and a reward function:

$$\pi(s): A(s) \rightarrow [0,1]$$

$$\forall_s \sum_{A(s)} \pi(s,a) = 1$$

$$Q^{\pi}(s,a) = \varepsilon_{\pi} \left\{ \sum_{k=0}^{\infty} \gamma^k \tau_{t+k+1} \mid s_t = s, a_t = a \right\}$$

$$Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t) \right]$$

$$\pi(s,a) = \begin{cases} \frac{ò}{|A(s)|} & \text{if } a \neq \arg\max_{a'} Q(s,a') \\ 1 - ò + \frac{ò}{|A(s)|} & \text{if } a = \arg\max_{a'} Q(s,a') \end{cases}$$

We write ò for the exploration factor, γ for discounting factor, α for learning factor, π for policy (ò-greedy), *s* for state, *a* for action, *t* for (discrete) time, $A(s)$ for action set, *Q(s,a)* for expected return in state *s* after action *a*, under current policy, *Q*(s,a)* for expected return in state *s* after action *a*, under optimal policy, and ε for expectation.

Case Study 2: Voice Command using supervised learning

The students applied their knowledge of neural networks, which is one flavor of supervised learning, to implement a simple speech recognition application. It took them one week to explore the operational principles of a basic neural network and to use this knowledge to design the application. For this application the Lego NXT Sound Sensor and the NXT brick were used to get the desired speech input. The Lego sound sensor is an envelope detector that measures the change in volume (amplitude of the sound signal) over time and not a real microphone. However, the envelope was sufficient input to build a recognition application that could distinguish between the words “Biertje” (beer) and “Champagne” (Champaign) by recognizing the difference in the word’s envelopes.

The NXT with help of its microphone recorded the words into sound samples that were then transferred via a Bluetooth connection to the computer. To make sure that the recognition was base on the difference in volume over time and not on the duration of the word, the length of both sound samples was equalized during the pre-processing. The sound samples were fed as input to the neural network that was created using Joone. The resulting output was then communicated back to the NXT that printed the results on its screen.

During the last two weeks of the course, the students were encouraged to create an extension pack for the Lego Mindstorms NXT set. These extension packs should empower other users in the Lego community to easily extend their Lego inventions far beyond what is possible with standard Lego. Two students decided to extend the neural network application that was build in the previous week.

The goal was to implement the neural network inside the NXT, so that it would no longer rely on a PC for its operation. Several possibilities were available to implement the neural network inside of the NXT brick. One option would have been to try to fit Joone inside the NXT brick. Although this would have been the most versatile solution, it would have moved the focus away from an understanding of neural networks towards a more in depth knowledge of the Java language. Therefore the students decided to build their own neural network from scratch inside the NXT. This allowed them to gain a better understanding of the formulas that describe a neural network and an in depth understanding of how to transform these formulas into Java code.

However, the NXT does not provide a user friendly graphical user interface (GUI) that would enable users to easily manage the recorded audio samples and the training process. The students therefore decided to create the Neural Network Manager software (see Figure 2) for the computer that performs the training of the neural network.

[figure]

Figure 2: Screenshot of the Neural Network Manager

Training the neural network on the NXT would in principle be possible as well, but of course at a much lower speed and only with an unfriendly user interface due to limitations of the NXT. It only has a small screen and four buttons to communicate with the user. A second reason for the preference of conducting the neural network training software is that the network itself is unlikely to stand by itself. Most likely, it would be integrated into other software. This software needs to be created on the computer anyway and hence there was little reason to renounce the use of a computer. Once the neural network is trained, it can be transferred back to the NXT. It can then be used as a standalone application or as a part in another program. The students were able to take advantage of their previous design education to create a highly useable GUI for the software. Hopefully, this will encourage other Lego users to take advantage of their software.

Neural Network Theory

We would like to conclude this case study by providing a short introduction to neural networks. This may allow readers that are not yet familiar with it to evaluate how much progress the students were able to make within three weeks.

Pattern recognition in general aims to classify data patterns extracted from raw data [1]. This is a very powerful tool to recognize classes of patterns where the raw data shows small variation or when the exact features are not known. In many cases this can be done by using statistical information about the patterns or linear mathematical functions. When the data becomes more complex as well as the segmentation of the different patterns, neural networks are very suitable to perform pattern recognition tasks (Haykin, 1999).

Neural networks in, for example, human brains consist of neurons connected trough synapses forming a complex network. Artificial neural networks feature layers of neurons. A simple neural network at least has an input layer with neurons, an output layer with neurons and at least one hidden layer. All the neurons in a layer are interconnected trough synapses to the next layer of neurons. Every neuron is connected to every neuron in the next layer (full synapses).

The synapses function as a weight factor and the neurons function as a mathematical function. Input can be fed into the neural network and is multiplied by the weight factors of the synapses. Neurons in the next layer apply a mathematical function, for example a sigmoid function to the sum of all the input values multiplied by their weight factor. This process repeats until the output neurons get a value. The output will return values that represent a specific pattern, at least when the weight factors are correct:

$$O_j = \frac{1}{1 + e^{-\sum_{i=1}^{m} x_i * w_{ij}}}$$

where,

O_j = output value of neuron j

m = neurons in previous layer

x_i = value of neuron i

w_{ij} = weight factor of the synapse between neuron i and neuron j

A common way to train a neural network is by means of backward propagation. Back propagation is a supervised learning method which means that a set of input values coupled to desired output are used to train the network. The back propagation algorithm calculates the errors signal by comparing the actual output with the desired output. It then uses the error signal to update the weights. This iterative process is repeated until the actual output approximates the desired output, the network is then trained:

$\delta_j = \left(t_j - O_j\right) \cdot O_j \cdot \left(1 - O_j\right)$, where,

δ_j = error signal for neuron j

t_j = desired output

O_j = actual output

$\delta_j = O_j \left(1 - O_j\right) \sum_k \delta_k \cdot w_{kj}$, where,

δ_j = error signal for neuron j

O_j = actual output

δ_k = error of (output) neuron k

w_{kj} = weight factor of synapse between neuron j and k

$w_{ij}(t+1) = w_{ij}(t) + \eta \cdot \delta_j \cdot O_i$, where,

$w_{ij}(t+1)$ = new weight for synapse between neuron i and j

$w_{ij}(t)$ = current weight for synapse between neuron i and j

η = learning rate

δ_j = error signal on output

Oi = input signal of synapse

Conclusions

We described the embodied intelligence method to teach machine learning to design students. By using a tangible embodiment as a platform for machine learning, the environment of the machine learning program does not need to simulated. But more importantly, the embodiment provides the student with a tangible tool to understand and interact with a learning system. Lego Mindstorms NXT is a good platform for this embodiment. The Lego system allows the students to quickly build a machine and thereby enables students to focus on the machine learning. In addition Lego NXT, provides a Java Virtual machine on which students can execute Java programs. Java is a widely used object-oriented programming language. The combination of the Lego construction system with the ease of Java on the NXT is a very low hurdle that even students that do not have an affinity toward technology can overcome.

Many of the students played with Lego during their childhood. This positively loaded memory might have lowered inner barriers that technophobic students might have build up. It might have allowed them to approach the course with a more open attitude and thereby increased the opportunity for learning. A second factor that might have had positive influence on the students is the behavior of the robots. The Crawler robot demonstrates that even simple learning behavior embodied in Lego has the power to create affection and empathy with human observers. This might have further motivated the students to experiment with the machine-learning program.

But the embodied intelligence method does not only offer advantages for less technophile students, but it also offers enough room for advanced development. Within only three days certain students were able to build and use neural networks. They then continued to build their own neural network program from scratch, utilizing on the theory they learned in the preceding week. In the end, they were able to create a neural network software that is user friendly enough for the general Lego enthusiast. As an example application, they build a voice command system, which enables the Lego NXT to operate as a stand alone voice controlled device. Again, we have to emphasize that these were neither computer science students nor mathematicians. These were design students that normally create artifacts.

Only by enabling design students to understand, use and develop machine-learning systems, we can ensure that they will be able to create truly intelligent systems, products, and related services. The embodied intelligence teaching method can help achieving this goal and our experiences suggest that it has the potential to significantly help students that do not have an inhering affinity towards technology.

References

Bartneck, C., & Jun, H. (2004). *Rapid Prototyping for Interactive Robots.* Paper presented at the 8th Conference on Intelligent Autonomous Systems (IAS-8), Amsterdam.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. New York: Springer.

Gasperi, M., Hurbain, P., & Hurbain, I. (2007). *Extreme NXT: Extending the LEGO MINDSTORMS NXT to the Next Level*. Berkeley: Apress.

Haykin, S. S. (1999). *Neural networks : a comprehensive foundation* (2nd ed.). Upper Saddle River, N.J.: Prentice Hall.

Nehmzow, U. (2003). *Mobile robotics : a practical introduction* (2nd ed.). London ; New York: Springer.
Solorzano, J. (2002). LeJos. from http://lejos.sourceforge.net/
Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning : an introduction.* Cambridge, Mass.: MIT Press.
Watkins, C. (1989). *Learning from Delayed Rewards.* Unpublished PhD Thesis, Cambridge University, Cambridge.