Interesting stuff for the Major project in hardware acceleration of machine learning algorithms

Get started with DNNs in GPUs <https://developer.nvidia.com/deep-learning>

Deep Learning courses <https://developer.nvidia.com/deep-learning-courses>

Accelerated computing GPU training <https://developer.nvidia.com/accelerated-computing-training>

Machine Learning NVIDIA papers and videos <http://www.nvidia.com/object/machine-learning.html>

cuDNN – NVIDIA Deep Learning library for GPU <https://developer.nvidia.com/cudnn>

Basic tutorials around FPGAs and products: <https://embeddedmicro.com/tutorials>

FPGA projects and info: <http://www.fpga4fun.com/>

Deep Learning info and reading list: <http://deeplearning.net/>

Xilinx boards and training: <http://www.xilinx.com/training/>

Deep Learning framework: <http://caffe.berkeleyvision.org/>

OpenCL - AMD “CUDA” AMD APU <http://developer.amd.com/tools-and-sdks/opencl-zone/>

Interesting topics  OpenCL and CUDA to build libraries for AI and Machine Learning acceleration (evolutionary algorithms, ANNs, optimisation algorithms…)

Massively parallel programming for quadratic assignment problem solving

**Ben**

Background in Computational Intelligence

Knows of CUDA and GPU programming (not in practice) and parallelising programming

Knows C programming at a low level

Willingness to supervise

Library of CI techniques  the optimisation bit is not the bottleneck, the model itself might be

* Optimisation GPGPU may be on the evaluation or better modelling
* Fuzzy logic can be quite demanding => parallel this? Probably the best candidate to optimise with GPGPU; GAs when evaluating generations

Two projects that could align:

1. Combine fuzzy logic and game theory  apply to intelligent transports (those two things) to improve decision making
2. SUMO, existing software, currently non-parallelised; simulator where each vehicle is being simulated in a large scenario; simulation of a traffic network. GPGPU could speed up the simulation.

Send an email with an abstract and a proposed title with my interests

Logan’s run; CI for intelligent traffic

DEEP LEARNING

Deep CNN architectures  <http://cs231n.github.io/convolutional-networks/>)

ResNet (Residual Networks) state of the art in CNNs

**General structure of the report**

Abstract

Introduction

* resurgence of machine learning thanks to massive parallel processing (GPGPU) to form Deep Learning
* importance of visual input in how humans interact with the world
* Convolutional neural networks to process images and detect features
* Problem statement: CNNs not have been widely used in continuous decision spaces. Training and optimising CNNs is complex and requires massive amounts of data -supervised. Could evolutionary strategies aid in the optimisation of CNNs for real time decision making?
* Overview of unsupervised training methods for deep learning → Reinforcement learning, Deep Q-network
* Overview of evolutionary strategies and algorithms → genetic algorithms, NEAT, Hyper-NEAT / ES-Hyper-NEAT
* Talk about RL policy search (EA, fitness based) vs value function search (TD, tune from examples)
* EA as an effective tool for RL tasks (Evolutionary computation for RL for reasons)
* Other neuroevolution of topologies methods: TWEANS (topology and weight evolving ANNS) 3 14 17 61 75
  + An evolutionary algorithm that constructs recurrent Nns
  + Neuroevolution: from architectures to learning
  + Evolving artificial neural networks (Yao)

Task

* ViZDoom framework, description of different scenarios
* Keras as a deep learning framework to use CNNs

Approach

* Using NEAT / Hyper-NEAT to evolve the topology and weight values of a CNN to solve various scenarios in ViZDoom
* Naive approach (direct NEAT) does not work → high dimensionality
* Dimensionality reduction by evolving a CNN that learns the most important features on an image (not compression)
* From features, NEAT / Hyper-NEAT is used to evolve a controller on different scenarios

Methodology

* Custom genetic algorithm to evolve Feature Detector → best candidate is then passed on to the next stage
  + EA for RL article as a basis to design it (phenotype-genotype mapping, mutation operators, EA as opposed to TD)
  + Image set is selected to show a representative collection of situations on each scenario (enemy in different locations, all items...)
  + Trained until convergence? (at the moment, 1000 generations)
  + Phenotype-genotype mapping is done to conserve phenotype similarity to genotype similarity, suggested for optimal EA (Holland, 1975 in EA for RL)
* NEAT / HyperNEAT then evolves the controller (input = features, output = actions)
  + To avoid memorising the scenario, sufficient randomness is introduced to the scenarios: custom (random position of the enemy), health gathering (initial position + random spawning of items), defend the centre (spawning of enemies is random), complex room (spawning of enemies), cig (spawning of enemies, initial player location).
  + Individual fitness calculated over sufficient episodes depending on the scenario (if the initial location is random, more than one scenario)
  + Partial rewards (shaping rewards) are used in each scenario to aid training (scenario-dependent) → tackling the credit assignment problem
    - Killing enemy, surviving time, picking up items
  + Trained until convergence? At the moment, stopped when it slows down
* Testing final performance → scenario dependent, but average score over 100 episodes

Experimentation (focus on only one weapon, Pistol)

* Series of experiments to determine the influence of certain design factors
* Feature detector:
  + Number of features (tie with controller experiments)
  + Network architecture
  + Size and nature of training image set (many random pics, selected few)
  + Evolution parameters (only if time allows it)
  + Plot fitness (assumption, higher diversity gives higher discerning capabilities later on)
* Controller:
  + Number of features (tie with FD experiments)
  + Different shaping rewards (shooting\_reward, poisoning over time)
  + Number of states (1 vs 3 vs 5)
  + NEAT vs ES-HyperNEAT
  + Output nature: 1 action = 1 state vs 1 action/axis = 1 output unit
* Compare performance with:
  + Supervised CNN
  + RL CNN

Discussion

Conclusions and further work

* Looking at co-evolution and distributed methods to evolve Fds and Controllers (Larranaga and Lozano, 2002; Hansen et al, 2003; Rubinstein and Kroese, 2004).
* Limitations of EA (EA for RL): online training, rarely visited states (sections of the genome that would react to them accummulate mutations as there is not evol pressure over them)
* Competing conventions problem (permutation problem) when evolving network topologies (crossover kills functionality)

**Introduction**

**Background**

The field of Machine Learning has been relaunched with the increase in popularity of Deep Learning. This has been possible for the increase in computational power, with massively-parallel processing leading the charge [1] [2], allowing researchers to explore more complex models for learning.

Convolutional Neural Networks have attracted recent attention, particularly in image recognition and classification [3] [4]. Part of its success is they are very efficient at reducing the complexity of feedforward fully-connected networks

Aside from being used on their own, CNNs have also been incorporated as part of a broader approach in Deep reinforcement learning, where they perform well approximating reward functions [5].

Continuing with the tradition of using games as a test bed for AI [6], recent times have seen an important milestone being reached: Go, a game well known for the complexity of its game space, saw the first computer to defeat the human world champion in a 5-games match [7]. AlphaGo guides its play using tree search in combination with Deep Networks –networks with multiple and complex hidden layers- to evaluate game nodes.

Moreover, the AI community is focusing more than ever on competitive events to develop algorithms to play games based on Deep Learning [8] [6]. One such competitions is ViZDoom [9], which restricts the inputs from the world exclusively to visual information from the rendered scene. CNNs and Deep RL are ideally placed for this type of scenarios.

**Aim**

The aim of this project is to use Deep Learning methods to design and implement a ViZDoom controller. Due to the restrictions of the framework, the agent will only receive information from the world in two forms: 1) visual data from the rendered screen; and 2) character data such as ammo, weapon selected, health status, etc. The agent will be required to perform actions such: 1) navigating through a 3D maze; 2) attack other agents, and 3) stay alive.

The focus will be ***to explore the suitability of Evolutionary algorithms for the optimisation of convolutional networks*** *-with particular interest on the NEAT algorithm* [10]- in a complex, real-time scenario ***based on visual input***.

**Relevant publications**

The foundation of this project is to use Neural Networks as a means to effectively interpret image data to control a game bot. In the literature this is commonly referred as *neurovisual control*. Authors have proposed neurovisual solutions for games such as Quake II [11] with moderate success in defined scenarios. The Quake II controller consists of a fully connected recurrent network that uses a retina-like input image –more resolution in the centre- to successfully learn how to find and kill enemies. One of the contributions of this paper is the semi-supervised learning process used to train the network, whereby a non-visual neural net is evolved first and then used to train the neurovisual network.

With the increase in popularity of Convolutional Networks, far more effective in the domain of image classification, authors have started using them to guide game play in different forms. Mnih et al [5] describe an enhancement of traditional reinforcement learning by using a convolutional network as its value function. The CNN uses raw pixel data as input, outputting an estimation of the future reward. The paper also proposes an efficient way of training such networks (DQN) and demonstrates it to be successful when playing a collection of Atari 2600 games, some to a level higher than humans.

Given the level of complexity of CNNs, its depth and the type of layers involved (convoluted, pooling / downsampling, fully-connected, etc), one of the key issues is how to create a network topology that optimises the target function. Although this has traditionally been tackled by handcrafting a layout [12], several authors have started to make use of automatic optimisation techniques to fine-tune entire networks, with evolutionary strategies at the forefront [13] [14].

The basis for this project is the Neuro Evolution of Augmented Topologies algorithm [10], an algorithm that uses an evolutionary process to optimise a neural network architecture. By automating the topology design, NEAT can efficiently produce highly optimised network layouts. The evolutionary process uses three types of mutation operators: 1) modify any of the network’s weights; 2) add or remove units to the network; and 3) add or remove connections between units.

Although NEAT was designed to optimise the topologies of feedforward networks, some derived evolutionary algorithms have already been applied in the domain of Deep Learning. Of particular interest is the work by Koutnik [14], which shows how neuroevolution is used to scale-up a network architecture used in reinforcement learning to create a driving policy in TORCS.

From a theoretical perspective, Maul and Bargiela [15] discuss neuroevolution opportunities for deep neural networks, comparing standard and cooperative evolutionary methods to propose general guidelines for the technique. Verbancsics and Harguess [16] suggest the combination of evolved networks using HyperNEAT (a modification of NEAT) with other Machine Learning techniques to improve performance.

**Objectives and deliverables**

1. To explore the suitability of Evolutionary algorithms for the optimisation of deep neural networks in a complex, real-time, large decision space scenario.
2. To use the findings to implement visual controllers to complete task-specific scenarios in ViZDoom.

**Intro from article**

The Open Racing Car Simulator (TORCS) has become an important tool for AI researchers interested in the development of autonomous driving strategies. Its pseudo-realistic simulation of racing events –physics, damage, engineering setup, etc.- together with the ease to design and implement new driving policies have contributed to its popularization, spawning multiple competitions where researchers submit their drivers to competitive tournaments [1] [2]. Previous successful submissions include Finite State modular approaches [3], fuzzy-based controllers [4], coevolution strategies for fine tuning car parameters [5] and feedforward neural networks that learn by imitation [6].

Most of the previous approaches use accurate track information –such as width of the current segment and distance to track limits-, opponent metrics –such as angle between our car and next driver- and any other data that is available in the TORCS environment. However, as humans we drive fundamentally based on local information available to us via our senses –mainly sight. Efforts to restrict the amount of data provided to drivers are seen in [1], though they still pre-process the information making drivers passive receptors of sensory information. The aim of this project is to investigate the feasibility of an autonomous driver that uses visual-only information to navigate tracks in TORCS.

Strategies that use visual input have been applied in a few scenarios: from neurovisual control of a Quake II bot [7] to an attempt to General Game Play for Atari videogames [8]. In the driving domain, one of the first implementations to use visual information as the main input for steering is ALVINN [9], a system for land vehicles that uses a 3 layer recurrent neural network to process input from a camera and a laser ranger to produce a recommended steering direction. In the TORCS environment, [10] showed how a recurrent neural network layout could be constructed using evolutionary algorithms to navigate –steer, accelerate and brake- a simple track using visual information. Although showing promise, these strategies are somewhat simplified in either the nature of the input –ALVINN uses distance sensors to aid visual input- or its complexity –simple network topologies, the TORCS driver performs only on a particular predesigned track. Those simplifications were imposed in no small part due to the high dimensionality nature of the visual input and the computational expense of processing it in real time. Nowadays those limitations are being overcome with the advent of massive parallel computation and the use of GPUs for General Purpose programming (GPGPU) [11] [12].

Taking advantage of the power of massive parallel computation, machine learning techniques have been brought to a new level in a domain called Deep Learning, including more complex models in Reinforcement learning, genetic algorithms and most of all neural networks. Deep neural network models have been shown to be ideal to process complex visual data [13], particularly in image classification [14] and pattern recognition [15] [16].

Convolutional Neural Networks (CNNs) are one such models; they consist of a series of partially-connected layers with local connectivity and shared weights to allow detecting patterns irrespective of their position in the image. CNNs are naturally good at feature extraction using multiple local maps that arise from the original image.

Recent advances in game AI can be attributed to the application of deep CNNs (with multiple layers) to decision making: deep reinforcement learning to play Atari games [18] and the newsworthy AlphaGo computer player that defeated the world champions [17] using deep neural networks as evaluation function on Go boards to predict their value in an attempt to shorten the tree search.