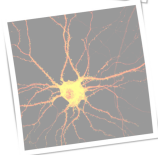
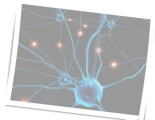
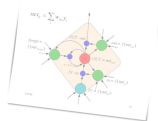
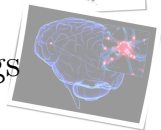


# Imitating Human Play from Game Logs (Workshop)

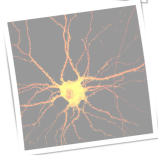
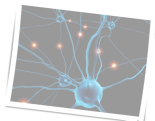
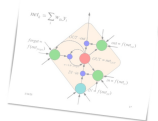
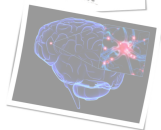
Spyros Samothrakis

July 20, 2015



# Imitating Human Play: A brief introduction

## The workshop



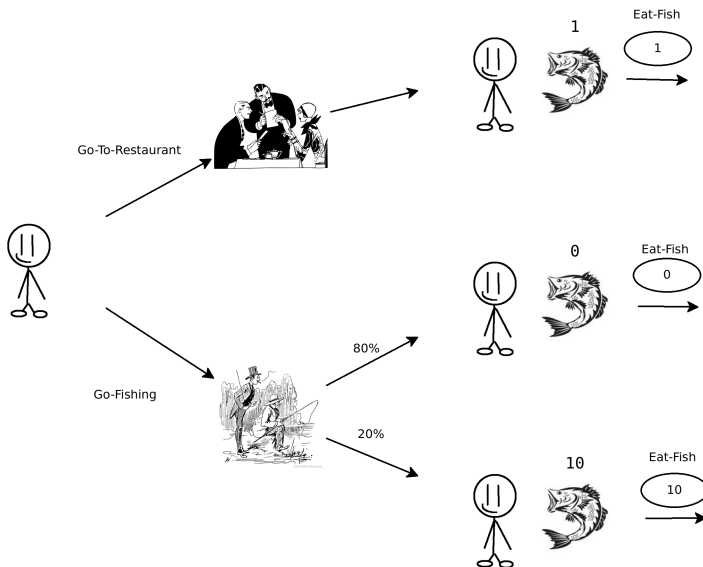
# GOALS

- ▶ Be able to explore the relevant literature independently.
- ▶ Gain enough experience in supervised learning methods in order to incorporate player behaviours in your game.
- ▶ What you will need:
  - ▶ A Laptop, almost everything tested in Mac/Linux, NOT windows
  - ▶ A working python installation
  - ▶ Clone this repo: <https://github.com/aigamedev/nuclai15>
  - ▶ Make sure you can run train.py (i.e. see which imports are failing and get them with pip)
  - ▶ Most important imports: **pandas, scikit-learn, scikit-neuralnetwork**

# THE PROBLEM

- ▶ Let's assume you have access to real game logs, i.e. data generated through human game-plays
- ▶ Is it possible to replicate, even partially, the observed behaviour of players?
- ▶ Why would anyone want to do that?
  - ▶ Copy player behaviour and incorporate into NPCs (i.e., more “human-like choices”).
  - ▶ Find optimal counter-strategies and incorporate into NPCs.
  - ▶ Bootstrap some other learning method.

# FISHING TOON: PICTORIAL DEPICTION



# EXPECTED REWARD

- ▶ In a sim like-game a player has to choose between two different actions
- ▶ Go-To-Restaurant or Go-Fishing

# THE MARKOV DECISION PROCESS

- ▶ The primary abstraction we are going to work with is the Markov Decision Process (MDP)
- ▶ Very rarely made explicit, though always there implicitly
- ▶ MDPs capture the dynamics of a mini-world/universe/environment/game
- ▶ An MDP is defined as a tuple  $\langle S, A, T, R, \gamma \rangle$  where:
  - ▶  $S, s \in S$  is a set of states
  - ▶  $A, a \in A$  is a set of actions
  - ▶  $R : S \times A, R(s, a)$  is a function that maps state-actions to rewards
  - ▶  $T : S \times S \times A$ , with  $T(s'|s, a)$  being the probability of an agent landing from state  $s$  to state  $s'$  after taking  $a$
  - ▶  $\gamma$  is a discount factor - the impact of time on rewards

## GOALS (A BIT MORE FORMALLY)

- ▶ Learn  $\pi(s, a)$ , a mapping between states and actions, called the *policy*
- ▶ Intuitively: “How should I act under this and this condition”?
- ▶ If we have logs from past games we can try to learn this mapping
- ▶ We have just restated our goal in more formal fashion - let's discuss a bit more



# REAL GAMES

- ▶ Real games often have a very large action and state spaces
- ▶ We can keep track of what players are doing
- ▶ ...but even for the simplest of cases learning a direct mapping is impossible
- ▶ Hence, the need to approximate (more on this later...)
- ▶ We will explore Linear Functions and Neural Networks in this workshop, but **many** more options are available

# INVERSE REINFORCEMENT LEARNING

- ▶ “Given that a player is acting optimally, can we copy her behaviour?”
- ▶ We are not going to cover this here, but it’s worth discussing it
- ▶ Why does it even make sense?
  - ▶ Is it more concise to learn  $R^*$  compared with  $\pi$ ?
  - ▶ Need for assumptions over the possible class of policies

# BEHAVIOURAL/SUPERVISED LEARNING

- ▶ Learn the sequences of actions using supervised learning
- ▶ Much simpler approach
- ▶ Obviously you have observed some sequence of  $(s_0, a_1 \dots s_n, a_n)$
- ▶ Learn a policy  $\pi$  from these actions directly
- ▶ This is what we are going to discuss in this workshop

# PREFERENCE LEARNING

- ▶ Supervised Learning
- ▶ One can learn to rank actions
- ▶ Is action  $a_0$  better than action  $a_1$  under state  $s$ ?
- ▶ Multiple methods of doing this, not to be covered here

# REINFORCEMENT LEARNING

- ▶ Not covered
  - ▶ ... but you can possibly treat action sequences as some kind of if implicit policy
  - ▶ ... and try to learn a value function
    - ▶ “What is the the average sum of expected rewards at state  $x$ ?”
- ▶ With off-policy learning you might event attempt to learn a better policy than the one actually being executed!
  - ▶ Search Q-Learning for more

# THE WORKSHOP

- ▶ Clone this repo: **<https://github.com/aigamedev/nuclai15>**
- ▶ Run `train.py`
- ▶ Make sure you you can run everything
- ▶ We now essentially have a supervised learning problem

# THE DATA PIPELINE

- ▶ Define the problem
- ▶ Data collection
- ▶ Data munging
- ▶ Metric selection
- ▶ Algorithm selection
- ▶ **Post-processing** (not covered)
- ▶ **Deployment** (not covered)
- ▶ **Experimental Evaluation** (not covered)
- ▶ (Above according to Microsoft research)

# PROBLEM DEFINITION

- ▶ Everything we discussed until now
- ▶ ...find what the agent is going to go next
- ▶ Define 3 actions per axis
- ▶ Concentrate ONLY on movement, i.e. where am I going to move next?
  - ▶ Re-create the Markov Property using 10 past (x,y) observations
  - ▶ Use other player positions
  - ▶ Let's see some code



# DATA COLLECTION

- ▶ You will need some kind of in-game probe
- ▶ Sensors on a player
- ▶ Logs from some source
- ▶ For DOTA there are online matches

# DATA MUNGING (1)

- ▶ Key tools: Pandas, numpy, scipy
- ▶ Create CSV for further pre-processing
- ▶ `extract_data_vectors.py`
- ▶ You can run it, but it requires a game log .dem file

# METRIC SELECTION - ALGORITHM SELECTION

- ▶ Inside train.py
- ▶ Let's have a look
- ▶ Mean Squared Error is our selected metric - but it's not the optimal

# FROM LINEAR REGRESSORS TO NEURAL NETWORKS

- ▶ Regressor in the form  $y_w(\theta) = w_0\theta_0 + w_1\theta_1 + \dots + w_k\theta_k + b$ 
  - ▶ Equivalently  $y_w(\theta) = \mathbf{w}\theta + \mathbf{b}$
  - ▶ Learn  $\mathbf{w}$
- ▶ Nested  $y_w(\theta) = \mathbf{w}^n(\max(0, \mathbf{w}^{n-1}(\max(0, \dots) + \mathbf{b}^{n-1}))) + \mathbf{b}^n$ 
  - ▶ Learn  $\mathbf{w}$

# MODERN TRICKS OF THE TRADE

- ▶ Better initialisation methods (e.g., unsupervised pre-training, Glorot initialisation)
- ▶ Better training methods (e.g., ADAM, RMSPROP)
- ▶ Better activation functions/units (e.g., Rectifiers, Maxout)
- ▶ Better regularisation methods (e.g., Dropout)
- ▶ Better weight sharing/convolutional layers (e.g., Fractional Max-pooling)
- ▶ Better hardware (GPUs)
- ▶ **Most of these have equivalents in `sklearn-neuralnetwork`**

# WORKSHOP TASKS

- ▶ Run experiments using a dummy classifier
- ▶ Run experiments using a Linear Classifier
- ▶ Run experiments using a Neural network
- ▶ Change the metric to “accuracy score”
- ▶ Each run should get you progressively better results
- ▶ Move some of the pre-processing to a different CSV file and load this (e.g., past examples)
- ▶ Change network architecture (e.g., number of neurons) and re-run the experiments
- ▶ Change train.py to save the classifier to a file (using python pickle)