Fido: A Universal Robot Control System Using Reinforcement Learning with Limited Feedback

Joshua Gruenstein Michael Truell

Control System Objectives

Fido was created to fulfill the following goals:

- **Trainability**: Allow both human and autonomous training and retraining rather than programming
- **Universality**: Run on any robot and perform any task, even without prior knowledge of the host
- **Performance**: Require few learning iterations and low latency for quick and efficient training

To achieve this we developed a novel machine learning algorithm utilizing wire-fitted Q-Learning with a neural network, a probabilistic action selection model, and history sampling. The control system could be trained at a rate **3 times** the industry standard, allowing practicality over traditional pre-programmed control systems.

System Overview

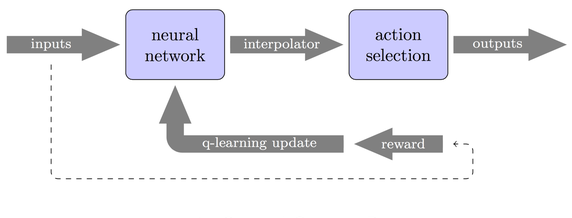

Figure 1: Control System Diagram

From a macro perspective, Fido can be viewed as a "black box" system, where a set of inputs (such as sensors) go in and a set of outputs (actions) go out. Fido's goal is to maximize given reward by altering its behavior. The system operates through the following steps:

1. Sensor values are fed to a neural network.
2. The neural network outputs data points, each is an action and its expected reward.
3. A wire-fitted least squares interpolator creates a continuous function of action to expected reward using these data points.
4. An action is chosen using an Softmax selection policy that dynamically adjusts Fido's exploration level as its confidence level changes.
5. After receiving reward, the neural network is trained to output a new set of data points using Adadelta for gradient descent. Dynamic sizing of the neural network and history sampling are employed to improve neural network performance.

Reinforcement Learning

- **Q-Learning:** Develops a function that intakes a state-action pair and outputs expected utility
- Ordinarily, the Q-function is modeled by storing state-action pairs in a table, but this is impractical for large state spaces
  - Use a function approximator instead to model the Q-function: **Artificial Neural Networks**
- With regular Q-Learning, no relation is made between similar states or actions
  - Can be optimized by coupling a wire-fitted interpolator with our neural network (**Wire-Fitted Q-Learning**)
- Complete solution: neural network is given sensor values and outputs data points on a graph of possible action vs expected utility, which are interpolated and used to generate a continuous function of action to expected reward.

Action Selection

- Cannot just perform the action with the greatest expected utility: must "explore" to be trainable and re-trainable
  - Use a Softmax Probability Distribution to select an action
- Cannot just hardcode the exploration level, Fido should stop exploring if confident in its actions and explore more if it is being retrained
  - Measure confidence using changes in the neural network weight matrix and adjust exploration accordingly

Training

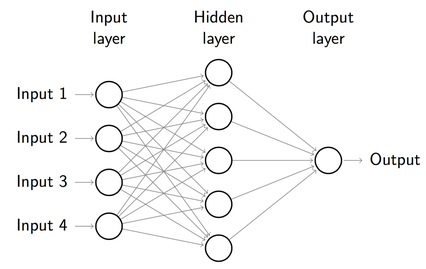

Figure 2: Single Output Feed-forward Neural Network

- Fido needs to update its model after receiving reward
  - Use Stochastic Gradient Descent to generate a new action to expected reward function for the past state
  - Train the neural network to output the data points that define this function
- Many neural network training algorithms require a learning rate that depends on the specifics of the task
  - Use the new Adadelta training method which dynamically adjusts learning rate, decreases latency, and results in better neural network accuracy
- **Control sequences:** Randomly train the neural network on some of Fido's past actions every reward iteration. Give greater weight to newer actions.
  - Allows training of action sequences for complex tasks
- Fido's optimal neural network architecture varies between tasks. Ex. larger networks are more suited for complicated tasks.
  - Efficiently grow and prune Fido's neural network using neuron sensitivity approximations gathered from fluctuations in the network's weights.

Implementation

Fido was programmed in C++, with no external dependencies. However, the simulator does use the SFML graphics library. The hardware implementation uses the Intel Edison embedded platform, a 3D printed chassis and a differential drive system.

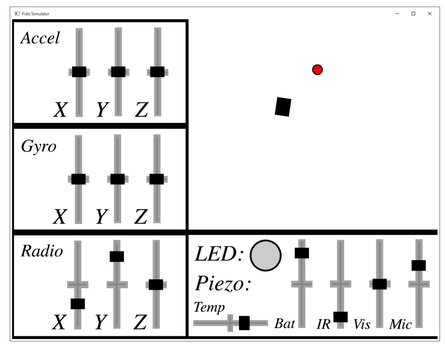

Figure 3: Fido Simulator Graphical User Interface

Applications

Support small businesses, rebuild the military.

Selected References

[1] S. Dini and M. Serrano, "Combining q-learning with artificial neural networks in an adaptive light seeking robot," 2012.

[2] C. Gaskett, D. Wettergreen, and A. Zelinsky, "Q-learning in continuous state and action spaces," pp. 417–428, 1999.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[4] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[5] R. Reed, "Pruning algorithms-a survey," *Neural Networks, IEEE Transactions on*, vol. 4, no. 5, pp. 740–747, 1993.

Results

Results were gathered both from simulation and hardware for a variety of tasks. Gathered data for each task included the number of learning iterations, or how many pieces of reward it took for Fido to master the task, action selection time, or latency in probabalistic action selection, and training time, or latency in updating Fido's model.

Table 1: Fido Results in Simulation

Task	Learning Iterations	Action Selection (ms)	Training Time (ms)
Flash	6	0.	6
Float to Point	14	1	6
Drive to Point	17	1	11
Line Following	18	0.	2
Noisy Line Following	21	0.	105

Table 2: Fido Results on Thing One

Task	Learning Iterations	Action Selection (ms)	Training Time (ms)
Stay Still	3	1	43.5
Drive to Point	18	4	65

Table 3: Fido Results on Thing Two

Task	Learning Iterations	Action Selection (ms)	Training Time (ms)
Drive Straight	13	2	30
Line Following	15	21	95
Fetch	8	1	70
Limping Line Following	6	20	37