Introduction

Tanks is a game that was originally created to accompany a section in the book "*Game Programming Gems 2*" that discussed the operation of the neural network in the game. In the original version of Tanks, an MLP neural network was used to set the elevation of the AI tank's barrel subject to a small random perturbation that was used to mimic random human-like errors. This rather simplistic and abstract model of aiming errors is replaced in this version of the game by a model of the actual random variations in the aiming of the game's author.

How to play

Left click in the area above your tank (the leftmost one) to indicate the direction you want to fire. Hit the enemy tank to score.

The player's and AI's scores are displayed in the top left and top right corners of the game window respectively. The arrow in the centre of the top of the window indicates wind speed and direction.

Hints

Click far from your tank to allow fine adjustments in barrel angle.
Do not move the mouse between shots because this prevents you making fine adjustments.
Take notice of changes in wind speed and direction.

How do I generate new training data?

Set `boGeneratingErrorTrainingData=TRUE` (in `TankDoc.cpp`) and compile.

The game will start in a special mode within which you have control of the AI tank and the AI has no turn. Every time you fire a shot, the game compares the angle you set the barrel to with the optimal angle (as estimated by the barrel-angle neural network) to derive the angular error in barrel angle. This quantity is saved to disk (in the file `NewExemplarData.txt` by default). Once enough data has been collected (i.e. several thousand shots) it should be possible to load the data into the game to produce a model of your own random aiming variation.

Make sure the first line of the exemplar file gives the number of examples it contains, followed by the number of inputs to the conditional distribution model (which is one, by default). Before producing a new model, you'll need to rename the newly created exemplar file to that used for training (called `AimingErrorExemplarData.txt` by default).

…and how do I produce a new aiming error model?

Set `boGeneratingErrorTrainingData=FALSE` and `boLoadTrainedErrorNetwork=FALSE` (in `TankDoc.cpp`) and recompile. The latter flag indicates that we do not have a pretrained model to load and a new one will have to be created.

When the program is run, it will automatically load the exemplar data and create and train new unconditional and conditional aiming error distribution models from it. Because of the location of the training loop, the game window will not appear until training is complete, but the progress of training can be monitored by opening the backup file (the default name `ErrorTrainingBackup.cdm` is given in `CConditionalDistribution.cpp`) in a text editor or watching the debug messages in the compiler's output window. Training may take several hours (the pretrained conditional distribution model

that is provided on the CD was trained over night). Once the distributions have been trained to within the specified error tolerance (`dErrorModelTerminationError` defined in `TankDoc.cpp`), the game begins.

How do I play the game normally again?

If you wish to use the new aiming error models in future, rename the training backup file – called `ErrorTrainingBackup.cdm` by default – to that which is loaded at the start of the game (the default filename `TrainedAimingErrorModel.cdm` is given in `TankDoc.cpp`) and ensure that `boLoadTrainedErrorNetwork=TRUE` and `boGeneratingErrorTrainingData=FALSE`.

Where does all the AI stuff happen?

The conditional aiming error distribution model is trained and saved in the constructor of `TanksDoc.cpp` when `boGeneratingErrorTrainingData=FALSE` and `boLoadTrainedErrorNetwork=FALSE`. If `boLoadTrainedErrorNetwork=TRUE`, the game attempts to load a saved model from the file `TrainedAimingErrorModel.cdm` and proceeds to train it if its performance doesn't meet the criterion for ending training (i.e. that its performance is worse than `dErrorModelTerminationError`). The unconditional aiming error distribution is never loaded or saved because its training process is so simple, that it can easily be re-trained every time the game is starts.

During the game, the `OnTimer` function `in TanksView.cpp` is called every few milliseconds. When `boGeneratingErrorTrainingData=FALSE` and it's the AI's turn to fire, `OnTimer` function calculates the inputs to the barrel angle neural network (the scaled *x*- and *y*-displacements of the player's and AI's tank, and scaled wind speed). It then passes a pointer to an array containing these to the network (in the argument of the network's `pdGetOutputs` function) and gets the network's estimate of the best barrel angle in the array pointed to by the return value.

Depending on whether the shot about to be made is the first of a new game or not affects how random variation is added to the aiming. If it is the first shot (`ulShotNumber` is equal to one), a random error in the AI tank's aiming is produced by extracting a single sample from the unconditional distribution of first shot aiming errors, by calling its `dGetOutputs` function. If it is not the first shot however, the error made on the previous shot is used to adjust the distribution of aiming errors by using the conditional distribution model. In this case, the random error is produced by extracting a sample from the conditional distribution model by calling its `dGetOutputs` function.

Alternatively, if `boGeneratingErrorTrainingData=TRUE`, the AI gets no turn because the player controls the AI tank. Every time the player takes a shot (which is handled in `OnLButtonUp` in `CTanksView.cpp`), the aiming error is computed as the difference between the optimal barrel angle given by the barrel angle neural network and that specified by the player. This information is stored in the world object and finally written to the exemplar file when the projectile is destroyed (e.g.. hits the landscape – as determined in the `TimeStep` function in `CWorld.cpp`). In the current implementation, only the number of shots since the start of the current game and the aiming errors made on the current and preceding shots are recorded.

`CMLP.h` and `CMLP.cpp` are the header and implementation files for the MLP neural network class, `CUnconditionalDistribution.h` and `CUnconditionalDistribution.cpp` are the files for the unconditional distribution class, and `CConditionalDistribution.h` and `CConditionalDistribution.cpp` for conditional distribution class.

But where does the barrel angle neural network come from?

As mentioned in the introduction, this version of the Tanks game is a development of an earlier version that accompanied the article "Using a neural network in a game: A concrete example" in the book "*Game Programming Gems 2*". That article described the development of the barrel angle neural network in great detail and the interested reader is strongly recommended to read that article to learn more about the practical application of neural networks. This implementation of the Tanks game assumes some familiarity with the basics of neural network design and focuses on the more advanced issues of modelling and reproducing random components of behaviour.

John Manslow
02/10/2001