Actor Oriented Development in Scala and Akka

with Neuro-Evolution Examples

Aaron Broad, BCS
aaron.broad_at_me.com

Developer @ Accreon Inc.,
Masters Student @ UNB

May 8th, 2013

Overview

1 Introduction
- About the Talk

2 Scala
- Why Scala?
- Scala IDEs
- Scala Build Tools
- Scala Testing
- Scala Tic-Tac-Toe

3 Akka
- Akka Actors
- Testing Actors

4 Scala/Akka Tic-Tac-Toe AI
- Artificial Neural Network
- Tic Tac Toe Task
- Weights-only Evolving Neural Network Population

Scala Langauge

- "Scala" is short for scalable
- A language created by Martin Odersky, the lead developer of the javac version that introduced generics
- Is very strongly typed
- Prefers an entirely functional style of programming, with single assignment variables, filter-map-reduce, etc.
- Allows object-oriented style of programming, with multiple assignment variables, imperative loops, etc.
- Compiles to JVM Bytecode
- Can call legacy Java libraries

Akka Framework

- Named for "Old Woman" mountain in Sweden
- A rough porting of Erlang Open Telecom Platform(OTP) to Scala
- Also has a Java API
- Programs are a self-healing tree of small single-threaded actors that send and receive static messages
- Actors can be freely distributed within a self-healing cluster of Akka nodes
- Makes for self-healing highly scalable applications

Artificial Neural Networks

- Artificial Neural Network(ANN) is a network computing model of transmuting an input vector into an output vector
- ANNs are turing complete, and universal approximators of functions
- ANNs are made up of only a few simple components:
  - input nodes: forward any number from their input to their one or more outputs
  - synapses: forward any number they receive from their input to their output, after having multiplied it by a weight
  - processing nodes: sum all of their inputs, apply the sigmoid function to that sum, and forward the result to all of their outputs

Neuro-Evolution

- The best way to develop ANNs to perform a particular task, assuming you can repeat the task ad infinitum with little consequence(cost in materials and lives), is to simulate natural evolution, until a "solution network" appears
- Neuro-evolution's highly parallel nature is an ideal application of Functional/Actor-oriented filter-map-reduce pattern
- While its size, in both space and time, make it ideal for Akka's distributed computing and self-healing fault tolerance

Go Functional

- Functional code is easier to write and maintain
  - Things go in, things come out, and always the usual suspects
- Functional code is made for our new parallel world
  - Machine/Network can easily parallelize things that have no side-effects
  - LISP has been doing map-reduce for 55 years
  - Was slow, on single-core isolated computers
  - Suddenly, now its the right way to code, and Google acts like they invented the...

Stay on the JVM

- Functional code was always the cleanest to reason about, but it was slow... So C
- When OO came along, few went to Smalltalk (what!? IDEs and a virtual machine!?), so C++
- A decade passes, and hmm.. this virtual machine and IDEs, maybe it could work, but Smalltalk syntax is unfamiliar, so Java
- People cling to their legacy code, and eschew unfamiliar syntax
- 2 decades of Java, and everyone wants to mainline a new language, but they don't want to rewrite their Java libraries

Familiar

- Lots of hot new JVM langauges are vying to fulfill the new langauge itch:
  - Scala, Clojure, JRuby, Groovy, etc
- However, Scala has some unique advantages when we look at the history of the mainstream:
- Object-oriented until you can give up your habit, if you ever do
- Doesn't force polish notation on you like Clojure:
  - If everyone was going to do this (+ (- (* 3 2 3))) they'd have done it by now
- Has more C++ like syntax, and less smalltalkish syntax like Groovy, JRuby

Scala IDEs

- Choices
  - Twitter, and may others simply use good old Emacs
  - Typesafe(Akk/Scala maintainers) promote ScalaIDE, a plugin for Eclipse
- Assume everyone can get on with Eclipse in this crowd, but will explain Emacs Install

Emac24 Install

1. Windows: http://ftp.gnu.org/gnu/emacs/windows/
2. Ubuntu
   1. sudo add-apt-repository ppa:cassou/emacs
   2. apt-get update
   3. apt-get install emacs
3. CentOS:
   1. Create repo file /etc/yum.repos.d/emacs.repo: https://gist.github.com/AaronTheApe/5540012
   2. yum update
   3. yum install emacs

ScalaMode2 Install

Add the following to your $HOME/.emacs file:
(require 'package) (add-to-list 'package-archives '("melpa" . "http://melpa.milkbox.net/packages/") t) (package-initialize)
(unless (package-installed-p 'scala-mode2)
(package-refresh-contents) (package-install 'scala-mode2))

Eclipse Juno + ScalaIDE Plugin

Alternatively:

1. Eclipse: http://eclipse.org/
2. Sclala-IDE Plugin: http://scala-ide.org/

Build Tools

- Choices, choices:
  - Simple Build Tool(SBT): As the name says, a simpler build tool than Maven
  - Maven: You might not want to learn a new build tool and a new langauge simultaneously

Simple Build Tool (SBT)

- Homepage: http://www.scala-sbt.org/
- Install is as simple as unzipping it somewhere in your $PATH
- Follows Maven file structure (/src/main/language/*, src/test/language/*) and repos
- Also supports flat file structure
- Powerful scala integration
- ScalaTest integration (test, test-only)
- Looping of many commands
  - ~ (a tilde) in front of any command will generally cause it to repeatedly fire (test over and over)

Maven

Alternatively, you can use Maven:

- Maven plugin:
  http://scala-tools.org/mvnsites/maven-scala-plugin/

ScalaTest

- Homepage: http://www.scalatest.org/
- Written by Bill Venners, one of the co-authors of the Scala Bible
- Support for many kinds of testing DSLs: TDD(FunSuite), BDD(FunSpec), Acceptance(FeatureSpec)
- Also supports tests already written in JUnit or TestNG

Language Features

- functions are first class citizens
- val or var variables (single-assignment or multipl-assignment)
- case classes (one-liner JVM data beans)
- match instead of switch (rich pattern matching case branching)
- immutable collections or mutable collections
- apply (define what an object called as a function does)
- functional filter-map-reduce or imperative loops

Object-Oriented Tic Tac Toe

Partial Solution... don't want to create bad habits:
OOTicTacToe.scala

Functional Tic Tac Toe

Port of RickHall2000's clojure functional tic tac toe:
FnTicTacToe.scala
Original Clojure Version:
https://github.com/rickhall2000/functional-tic-tac-toe.git

Limits of Pure Functional

- Scala lets us program functionally or imperatively
- Functional is cleaner, easier to test, easier to debug, and avoids OO-type defects
- However, pure functions can't easily operate in distributed environments
- Requires anticipating and functionally handling myriad of distributed type errors
- Also, some problems naturally lend themselves to message passing
- Enter actors...

Akka Actors

- Single-threaded
- Has only one method
  - private receive(Message: Any)
- No shared data
- Receives static messages from Mailbox
- Sends messages to other Actors

Messages

- Actors can send messages to one another, but not directly
  - (Remember: receive is private, and further still, only the ActorSystem has its JVM object ref)
- Each Actor is given a unique ActorRef which is simply a unique URI for that actor in this system
- Actors can send messages to the ActorRef of another Actor
- In order message delivery is guaranteed between any two actors
- However, no global ordering of messages by time sent is attempted

Mailbox

- All sent Messages are delivered first to the actor's system assigned Mailbox, addressed by ActorRef
- The actor's Mailbox invokes an Actor's receive method and passes it a message
- The Mailbox will not invoke receive again until the first invocation returns
- The Mailbox will repeatdly invoke the actor's receive until the the Mailbox is empty
- If for any reason, and Actor cannot process a message, it dies, and the message is returned to the Mailbox

Supervisors

- Every Actor has a Supervisor(Super)
- First Actor created has special actor "Guard" as Super
- All other Actors have whichever Actor created them as Super
- Super is informed if any of its "Child" Actors dies and executes its Supervision Strategy
- If, the Super decides to restart the Child, a clone of the dead Actor can begin processing the Mailbox
- If the original Actor message processing had no side-effects, nobody will ever know, and no data has been lost

Supervision Strategy

- Default SupervisionStrategy is restart(one-for-one) on Exception, escalate on Throwable, stop on Kill or Init Failure
- Other predefined or custom SupervisionStrategies can be mixed in or defined:
  - One-for-many (restart all children when one fials)
  - Restart X times
  - Rerstart X times if reason for death is Y, always otherwise
- Escalate simply means the supervisor itself dies

Fail-Fast Engineering

- In Erlang/Akka Try/Catch inline Exception handling is possible but discouraged
  - Why so confident you can fix the problem?
    - Maybe, you don't posesse the required information?
    - Its hardware failure you can't code around, and continuued operation is pumping garbage everywhere
  - Why recover? You don't have any state to lose.
- Instead, structure the problem as top-down hierarchy of the smallest functional components
- If you even have to think whether it should be a separate Actor, make it one
- Fail-Fast on any exception, supervisor will give you another crack at it, or escalate

FSMs

- Syntax sugar, and proper design pattern enforcement, for coding of the receive method
- Standard Actor receive method simply matches message signature to an ordered list of patterns and handling functions for those patterns
- FSM enforces that an Actor has specific StateName types and specific StateData types
- And that in any given StateName, an Event(Message, StateData) maps to a new StateName with new StateData prior to the handling of the next Event

Teskit

- Akka provides akka.testkit package for testing actors
- Main components are:
  - TestProbe -> Object that can pretend to be any Actor, in order to test what an Actor does when it receives a particular message, or if it sends a message to the right place
  - TestActorRef/TestFSMRef -> Object that can wrap a real Actor, and allow the programmer to see/modify its internal state

Input Node

- Has one Input
- Has one or more Outputs
- Broadcasts all Stims from Input to all Outputs

Synapse

- Has one Input
- Has a Weight
- Has one Output
- Forwards any Stims from Input to Output with Weight applied

Processing Node

- Has one or more Inputs
- Has one or more Outputs
- Waits for one Stim message from each Input
- Applies Sigmoid Function to Sum of Stims: $1/(1+e^{-x})$
- Forwards result of function to all Outputs

Standard Two-layer Network

- The Input Vector is lined up with Input Nodes and each Input Node is fed one Stim Value
- All Inputs have a Synapse connecting them each to all "Hidden" layer nodes
- Hidden nodes are all Processing Nodes
- All Hidden nodes have a Synapse connecting them to all of the "Output" nodes
- Output Nodes are all Processing Nodes that are lined up with the Output Vector
- Each Output Vector is made from one Stim Value from the respective output nodes they are lined up with

Properties of Standard Two-layer Network

- Cybenko(1989): "Approximation by Superpositions of a Sigmoidal Function*"
  - A universal approximator of functions (Turing Complete)
- Fogel(1990): "Evolving Neural Networks"
  - The best solution network to any task can be found by simulating natural evolution on a population of Networks:
    - Create a random population that varies uniformly in the weights of their Synapses
    - Have each network perform the task X times
    - Allow the most successful ones to reproduce, while removing the least successful
    - Repeat until a solution network is discovered

Functional Tic Tac Toe

- See src/one/task/

Fogel Method

Weights-only Evolving Neural Network Population

- See src/one/method/fogel/

Learning More Scala

- Online Resources:
  - Twitter's Scala School: http://twitter.github.io/scala_school/
  - Free Online Graduate Course: https://www.coursera.org/course/progfun
- Offline Books:
  - Programming Scala: http://www.artima.com/shop/programming_in_scala_2ed
  - Scala in Action: http://www.manning.com/raychaudhuri/

Questions?