Functional Programming Languages for AI: Clojure

Efficient Programming

Christian Willner
christian.willner@studium.uni-hamburg.de

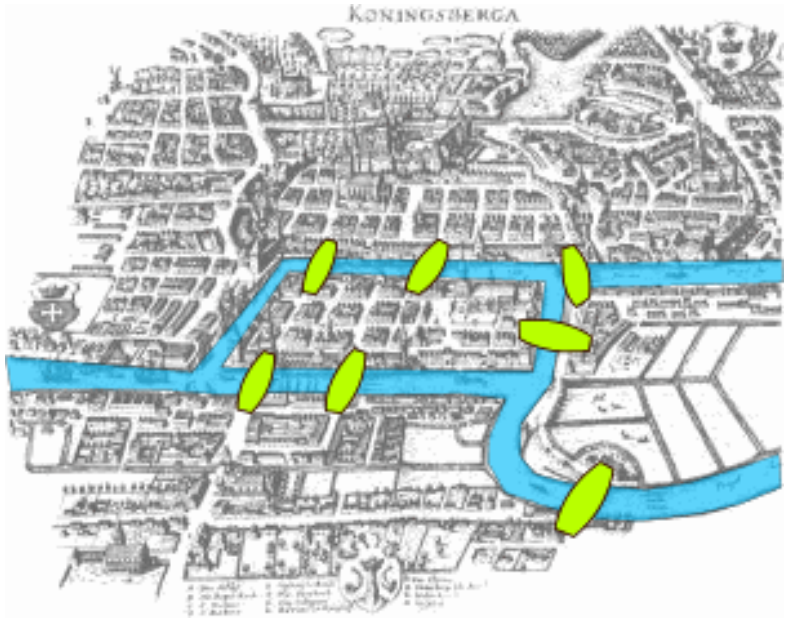
University of Hamburg
Dept. of Informatics

16.02.2021

(Overlapping) Fields of AI research

Fundamentals in AI research

Statistics and probability
Linguistics
Arithmetics
Numerical optimization
Graph theory



Seven bridges of Königsberg Problem[1]

[1] https://en.wikipedia.org/wiki/Seven_Bridges_of_Königsberg

Machine learning prerequisites

Machine learning:

- Fast linear algebra functions
- Space efficient data handling
- Concurrency

Data processing:

- Simple input transformation
- Short feedback loops

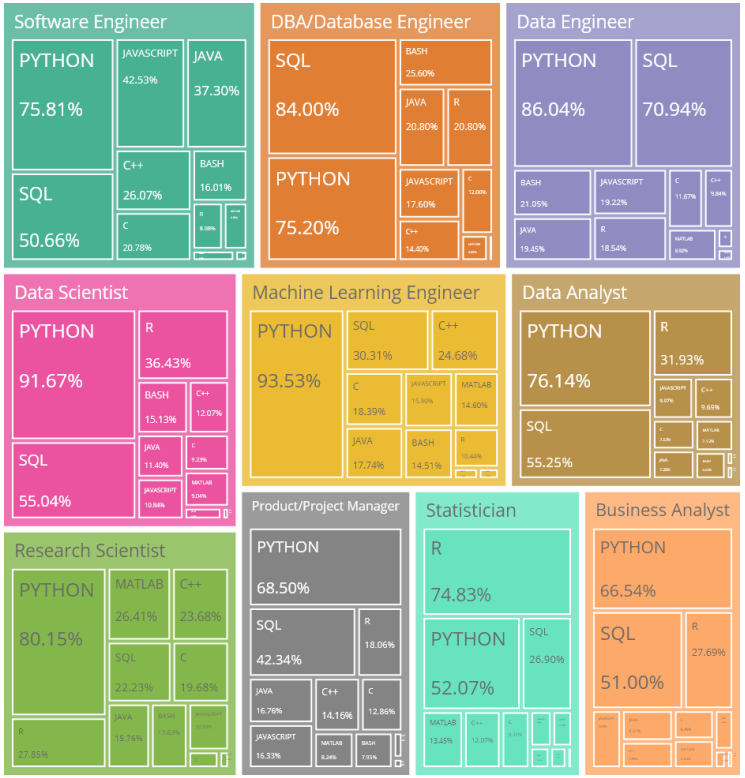
Ecosystem:

- Library supply
- Community support

Common languages in AI research [4]

Python, R,
Lisp, Prolog,
Java, C++,
Julia, SQL

(with FP patterns)



2020 Kaggle ML survey results[3]

[3] https://www.kaggle.com/carloseduardosilvabh/kaggle-2020-ml-survey-analysis

[4] https://www.techindiatoday.com/programming-languages-for-artificial-intelligence/

Functional programming languages towards AI

Functional programming

What is functional programming?

Programming paradigm

- Collection of concepts, patterns, standards, . . .

Functions as first class citizens

- Just like any other variable (e.g. argument or return)

Pure functions

- Referential transparency: no side–effects
- Replacements at compile-time

→ Consequence: Immutability

Functional programming for AI?

Immutable data

- Concurrency

Pure functions

- Easily swapped

Stateless

- Convenient for reasoning, testing

Higher abstractions

- Easier to understand and debug

Lisp for AI?

Semantics

- Convenient to prove computation properties[5]

Reprogrammable

- Compile, read and run at runtime, compiletime or readtime

Data representation

- One language: data, code and meta-programming

[5] Winskel is (almost) Right - Tobias Nipkow

Functional programming languages towards AI

Language focus: Clojure

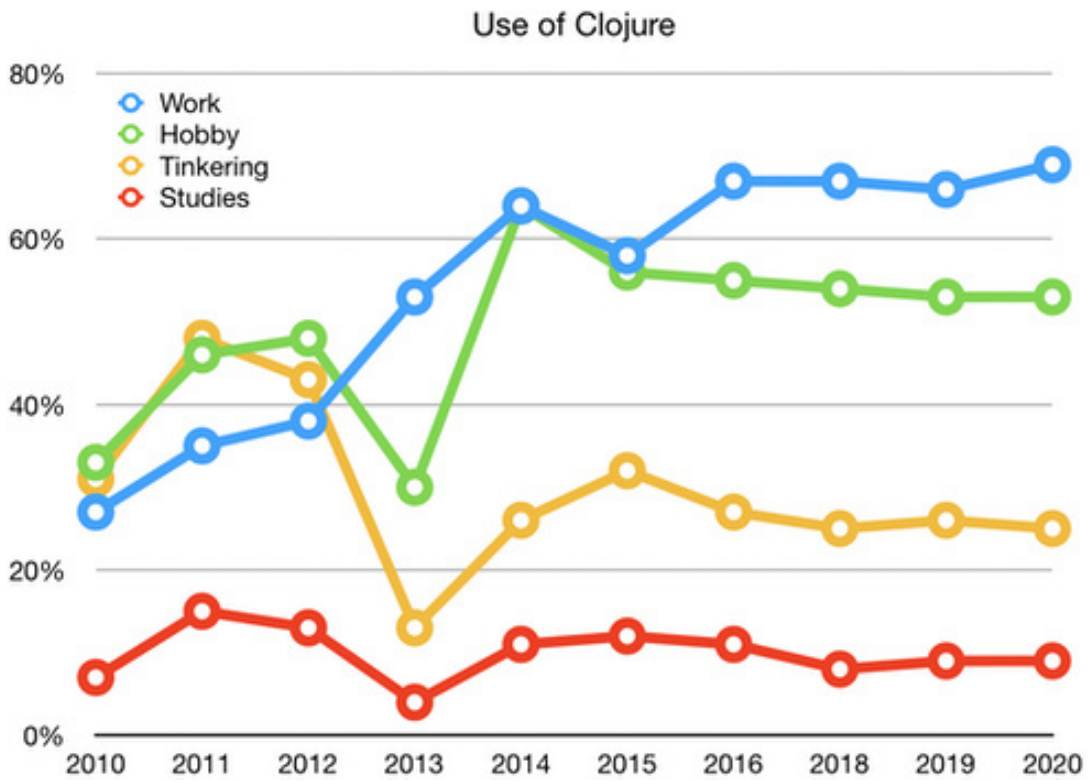
What is Clojure?

- Initial release 2007
  - A functional Lisp dialect for concurrency
- Very stable core
- Current version 1.10.1 (June 2019)
  - Community driven development

Statistics for the Clojure repository at GitHub[6]

[6] https://github.com/clojure/clojure

Use of Clojure



State of Clojure community survey 2020[7]

Business users are e.g. Apple, Netflix, Walmart & NASA

[7] https://de.surveymonkey.com/results/SM-CDBF7CYT7/

Interesting aspects for AI research

Hosted language

Homoiconicity & macros

Precise, consistent syntax

The REPL

Immutability as default

Dynamic polymorphism

Hosted language

JVM as primary platform for Clojure

- Base for interop with Java (JS for Clojure Script)

Other platforms:

- GraalVM (reduced start-up time $< 10\,ms$) [8]
- .NET
- Perl, Python, Erlang, C++11, Go, Ruby (to a lesser extend)

[8] clojureD 2020 - Michiel Borkent on "Babashka and Small Clojure Interpreter: Clojure in new contexts"

Homoiconicity

Code is data $\rightleftarrows$ data is code

- $\rightarrow$ Quote (treating code as data)
- $\rightarrow$ Eval (executing Data as code)
- Easy metaprogramming e.g. macros within the language
- Symbolic expressions

Example: code ⇄ data[9]

```
(html
 (head
  (title "The Title"))
 (body
  (h1 "The Headline" :class "headline")
  (p "Some text here" :id "content")))
```

Clojure code/ data

```
<html>
  <head>
    <title>The title</title>
  </head>
  <body>
    <h1 class="headline">The Headline</h1>
    <p id="contents">Some text here</p>
  </body>
</html>
```

HTML code

[9] https://stackoverflow.com/a/108102/4919081

Potential of macros[10]

'If you give someone Fortran, he has Fortran.
If you give someone Lisp, he has any language he pleases.'

– Guy L. Steele, co-author of the Java spec

[10] History of Programming Languages II,1996

Macros

- Tool to create syntax, useful for DSL
- Expanded at compile-time, evaluated at run-time
- Macros construct symbolic expressions, not string generation

```
(def macro unless
  "Similar to if but negates the condition"
  [condition &forms]
  `(if (not~ condition)
     ~@forms))
```

REPL[11]

Read Evaluate Print Loop

- Read: input stream $\rightarrow$ data
- Eval: data $\rightarrow$ data
- Print: data $\rightarrow$ output stream

- Not just a shell, immediate feedback, dynamic development
- Compiled on the fly, not interpreted
- Can be attached to running programs, even remotely
- Can be customized inside the Repl
  (custom error msgs, debugging, etc)

[11] Chicago Clojure 2017 - Stuart Halloway on Repl Driven Development

Functional programming languages towards AI

Syntax, overview & demo

Syntax [12] - data structures

```
(1 2 3 4) ; list
["a" "b" "c"] ; vector
{:place "Hamburg", :prefix 040} ; map
#{3 2 1 2 3} ; set
```

[12] Clojure in a nutshell by James Trunk

Syntax - functions & special forms

```
(function arg-1 arg-2 arg-3)

; examples:
(+ 1 2 3) ;-> 6
(max 1 2 3) ;-> 3
(filter odd? [1 2 3 4]) ;-> (1 3)
(if (even? 4) "yes" "no") ;-> "yes"
```

Overview - immutability

```
(def food [:cake :pie])
(conj food :cookies) ; -> [:cake :pie :cookies]
food ;-> [:cake :pie]
```

Syntax - defining functions

```
(def hello (fn [name] (str "Hello " name)))
(defn hi [name] (str "Hi " name))

(hello "Jacky") ;-> "Hello Jacky"
(hi "Juan") ;-> "Hi Juan"
(#(str "Holá " %) "Gina") ;-> "Holá Gina"
```

Overview - handling parentheses

```
(filter odd?(map inc(range 5))) ;-> (1 3 5)
; threading
(->> (range 5) ;-> (0 1 2 3 4)
     (map inc ,) ;-> (1 2 3 4 5)
     (filter odd? ,)) ;-> (1 3 5)
```

Demo - loading a book

```clojure
(ns nutshell
  (:require [clojure.string :as str]))

(def book (slurp "http://www.gutenberg.org/files/2701/2701-0.txt"))
(def words (re-seq #"[\w|']+" book))
(count words)
;; => 222685
(take 7 words)
;; => ("The" "Project" "Gutenberg" "EBook" "of" "Moby" "Dick")
```

Demo - top 10 words

```clojure

;; 10 most frequent words
(def most-common (set (clojure.string/split-lines
                        (slurp "./data/most-common-words.txt"))))
(->> words
     (map clojure.string/lower-case)
     (remove most-common)
     (frequencies)
     (sort-by val)
     (take-last 10)
     reverse)
;; => (["whale" 1230]
;;     ["upon" 568]
;;     ["ship" 519]
;;     ["ahab" 512]
;;     ["ye" 472]
;;     ["sea" 455]
;;     ["though" 384]
;;     ["head" 348]
;;     ["boat" 334]
;;     ["long" 332])

```

Demo - longest words

```
;; 10 longest words
(->> words
     distinct
     (sort-by count)
     (take-last 10)
     (group-by count)
     reverse)
;; => ([20 ["uninterpenetratingly"]]
;;     [18 ["characteristically"]]
;;     [17
;;      ["cannibalistically"
;;       "circumnavigations"
;;       "superstitiousness"
;;       "comprehensiveness"
;;       "preternaturalness"
;;       "indispensableness"
;;       "uncompromisedness"
;;       "subterraneousness"]])
```

Demo - longest palindrome

```clojure
;; longest palindrome
(defn palindrome? [word]
  (= (seq word) (reverse word)))

(palindrome? "racecar");; => true
(palindrome? [1 3 1]);; => true

(defn longest-palindrome [words]
  (->> words
       distinct
       (filter palindrome?)
       (sort-by count)
       last))

(longest-palindrome words)
;; => "deified"
```

Functional programming languages towards AI

Concurrency

Concurrency in general

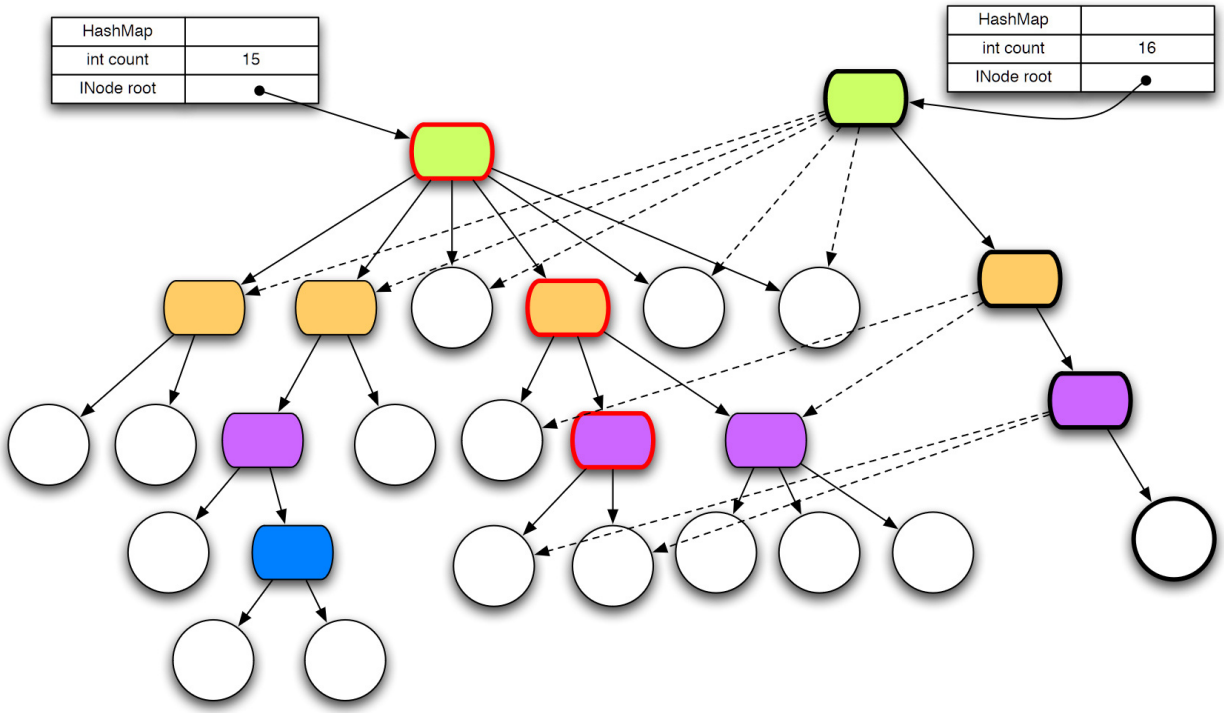
Locks can cause deadlocks

- Difficult to implement correctly
- Keeping track of locks is not enforced in Java / C#

Control is facilitated in CLJ ($\neq$ parallel language)

Solution: persistent data structures [13]

✓ Performance guaranteed through structural sharing



[13] Rich Hickey - Concurrency Support `https://youtu.be/dGVqrGmwOAw`

Clojure references

- References are the only mutable types in clojure
- Snapshot system (multiversion concurrency control)
- Commutative transactions supported

Usage:

- Vars
  (Within threads)
- Refs
  (Sync., between threads, automatic retry, no side-effects)
- Agents
  (Async., between threads, queue system, awaits, with side-effects)

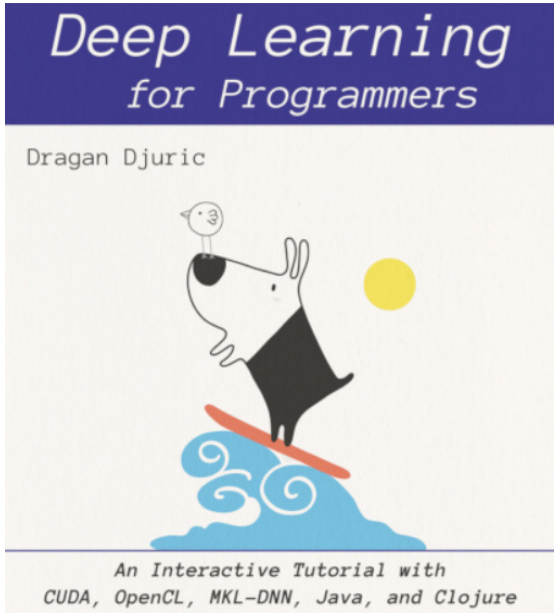
Performance of Clojure

"Neanderthal is also as fast or faster than the best Java and Scala native BLAS[14] wrappers" [15]

"[Deep Diamond is m]uch faster than Keras and TensorFlow on the GPU" [16]

Possible to use libraries from:

- Python (libpython-clj)
- Julia (libjulia-clj)
- R (R-clj)
- Ruby (zweikopf)



Cover of Deep Learning - Dragan Djuric[17]

[14] Basic Linear Algebra Subprograms

[15] https://neanderthal.uncomplicate.org/articles/benchmarks.html

[16] tps://dragan.rocks/articles/20/Going-faster-than-Tensorflow-on-GPU-with-Clojure

[17] tps://aiprobook.com/deep-learning-for-programmers/

Examples

GAN network

FLAN from GAN[20]

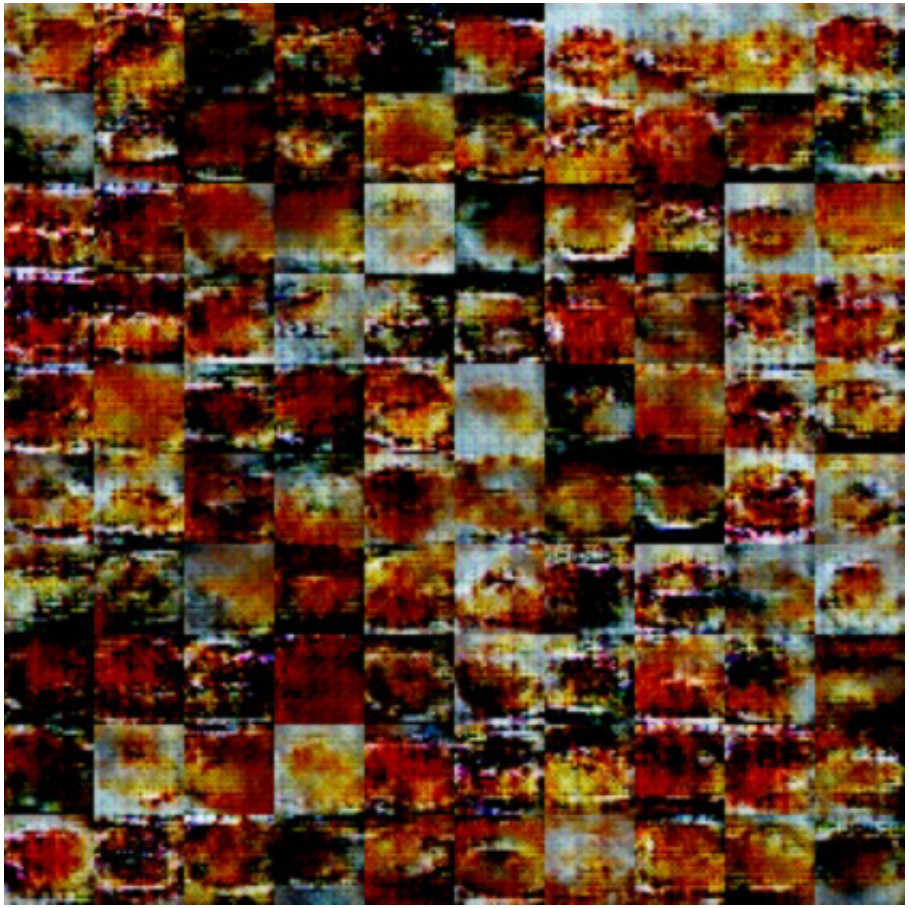
Using a generative adversarial network to create flan pictures with MXNet[18] [19]

[18] Deep learning library: `https://mxnet.apache.org/`

[19] MXNet now via: DJL (`https://djl.ai/`)

[20] http://gigasquidsoftware.com/blog/2018/12/18/how-to-gan-a-flan/

Flan database



Sample of scrapping 1000–5000 pictures of flans

Discriminator definition

```
(defn discriminator []
  (as-> (sym/variable "data") data
    (sym/convolution "d1" {:data data
                           :kernel [4 4]
                           :pad [3 3]
                           :stride [2 2]
                           :num-filter ndf
                           :no-bias true})
    (sym/batch-norm "dbn1" {:data data :fix-gamma true :eps eps})
    (sym/leaky-re-lu "dact1" {:data data :act-type "leaky" :slope 0.2})

  ...
```

Fake or real flan per convolutional neural netowrk

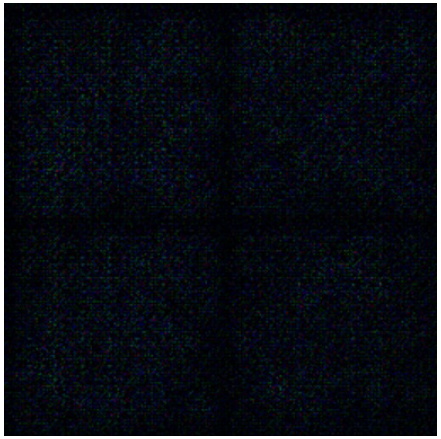
Generator definition

```
(defn generator []
  (as-> (sym/variable "rand") data
    (sym/deconvolution "g1" {:data data
                             :kernel [4 4]
                             :pad [0 0]
                             :stride [1 1]
                             :num-filter
                             (* 4 ndf) :no-bias true})
    (sym/batch-norm "gbn1" {:data data :fix-gamma true :eps eps})
    (sym/activation "gact1" {:data data :act-type "relu"})

  ...

```

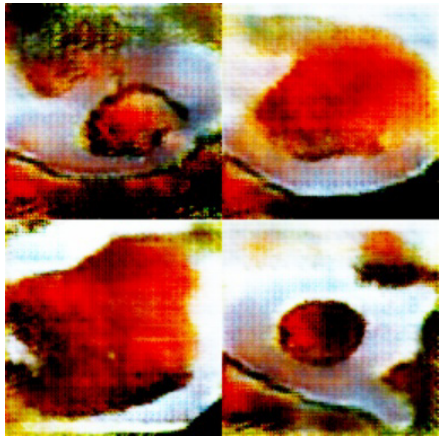
From random noise to flan with a deconvolutional network

GAN training progress I

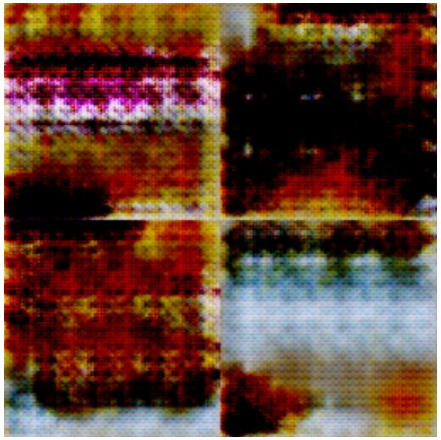
Epoch 0



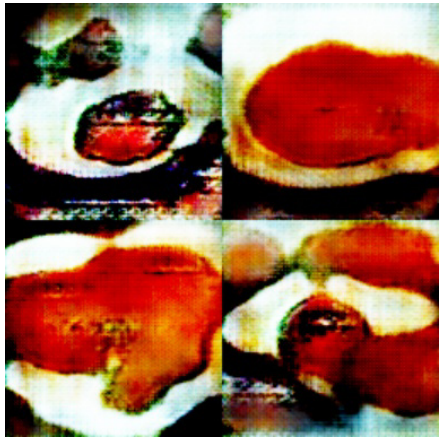
Epoch 23



Epoch 10

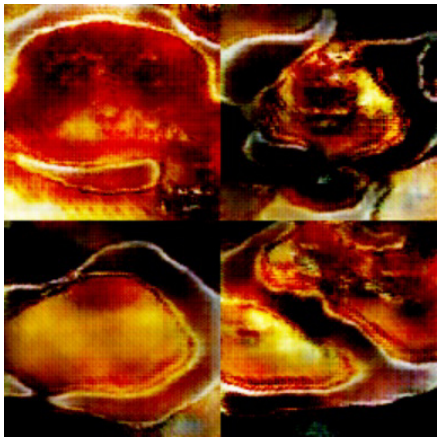


Epoch 33

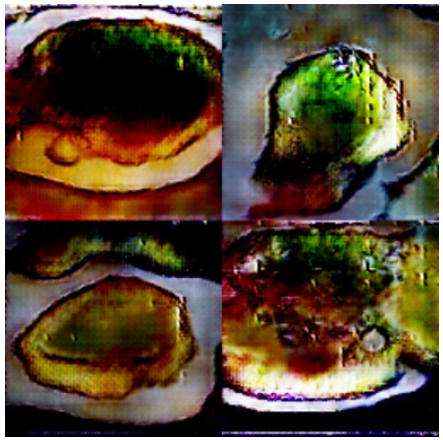


GAN training progress II

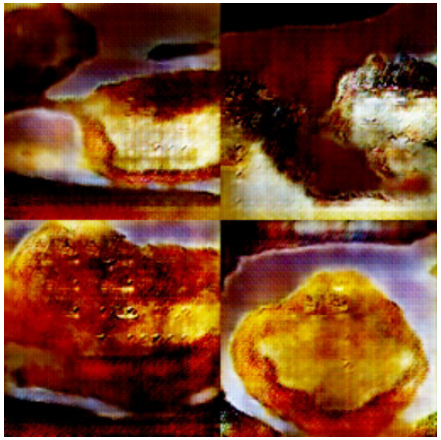
Epoch 68



Epoch 170



Epoch 161



Epoch 195

Examples

Genetic programming

Spec library - define, validate, generate data

```clojure
(ns spec-demo
  (:require [clojure.spec.alpha :as s])
  (:require [clojure.spec.gen.alpha :as gen]))

(s/def ::big-even (s/and int? even? #(> % 1000)))
(s/valid? ::big-even :foo) ;; => false
(s/valid? ::big-even 100000) ;; => true

(s/explain ::big-even 5)
;; => 5 - failed: even? spec: :spec-demo/big-even

(s/def ::vnum3 (s/coll-of number? :kind vector? :count 3 :distinct true ))

(gen/generate (s/gen ::big-even))
;; => 14218866
(gen/generate (s/gen ::vnum3))
;; => [-0.05636344105005264 1.381591796875 0.059814453125]
```

Spec library - define functions & exercises

```
(defn my-poly-3 [x a b c] (->> (+ x a) (* x) (+ b) (* x) (+ c)))

(s/fdef my-poly-3
        :args (s/cat :x number? :a number? :b  number? :c  number?)
        :ret number?)

(s/exercise-fn `my-poly-3 )
;; => ([(0 0 0.5 0) 0.0]
;;     [(-1 -1 0.5 -1.0) -3.5]
;;     [(-2.5 -2 -0.5 -1) -27.875]
;;     [(0.78125 -1 0.5 0) 0.257110595703125]
;;     [(-1 1.0625 0 -1) -0.9375]
;;     [(-2 -0.8125 -0.9375 0) -9.375]
;;     [(-1.96875 2.0 2 0.5) -3.316375732421875]
;;     [(-0.734375 3.1875 -3.4375 1.0) 4.847400665283203]
;;     [(1.0 2 -12 12) 3.0]
;;     [(19 -3 1.88671875 12) 5823.84765625])
```

Genetic programming scheme

→ Generate valid input data

→ Take samples

→ Evaluate

↻ New generation

[21]

[21] http://www.genetic-programming.com/coursemainpage.html

Genetic programming with spec[22]

```
(defn discriminator []
  (as-> (sym/variable "data") data
    (sym/convolution "d1" {:data data
                           :kernel [4 4]
                           :pad [3 3]
                           :stride [2 2]
                           :num-filter ndf
                           :no-bias true})
    (sym/batch-norm "dbn1" {:data data :fix-gamma true :eps eps})
    (sym/leaky-re-lu "dact1" {:data data :act-type "leaky" :slope 0.2})

  ...
```

Flan discriminator

```
(s/def ::layer (s/or :conv ::convolutional
                     :maxp ::max-pooling
                     :drop ::dropout
                     :mutld ::multiplicative-dropout
                     :relu ::relu
                     :prelu ::prelu
                     :bn ::batch-norm
                     :lin ::linear
                     :lrn ::local-response-norm))

(s/def ::layers (s/coll-of ::layer :min-count 1 :max-count 4))
```

Spec definition

[22] Carin Meier - "Deep Learning Needs Clojure" (Conj 2017)

Wrap-up

Summary

Machine learning capabilities

Machine learning:

- Fast linear algebra functions ✓ neanderthal, deep–diamond
- Space efficient data handling memory ✓ software transactional
- Concurrency ✓ structural sharing

Data processing:

- Simple input transformation ✓ demo
- Short feedback loops ✓ REPL

Ecosystem:

- Library supply ✓ interop
- Community support

Wrap-up

References

Where to start?

- Website: `clojure.org`
- Documentation: `clojuredocs.org`
- Slack: `clojurians.slack.com`
- Zulip: `clojurians.zulipchat.com`
- SciClj: `https://scicloj.github.io`

How to learn Clojure?

Books:

- Programming Clojure - A. Miller, S. Halloway, A. Bedra
- Clojure for the Brave and True - D. Higginbotham
- Living Clojure - C. Meier
- The Joy of Clojure - M. Fogus, C. Houser

Online–learning:

- `clojurescriptkoans.com`
- `4clojure.com`
- `braveclojure.com`
- `kimh.github.io/clojure-by-example`
- `exercism.io`

Sources I

[1] https://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg

[2] https://www.kaggle.com/carloseduardosilvabh/kaggle-2020-ml-survey-analysis

[3] https://www.techindiatoday.com/programming-languages-for-artificial-intelligence

[4] Nipkow T. (1996) Winskel is (almost) right. In: Chandru V., Vinay V. (eds) Foundations of Software Technology and Theoretical Computer Science. FSTTCS 1996. Lecture Notes in Computer Science, vol 1180. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-62034-6_48

[5] https://github.com/clojure/clojure

Sources II

[6] https://de.surveymonkey.com/results/SM-CDBF7CYT7/

[7] clojureD 2020 - Michiel Borkent on "Babashka and Small Clojure Interpreter: Clojure in new contexts

[8] https://stackoverflow.com/a/108102/4919081

[9] Bergin T., Gibson R. (1996) History of Programming Languages II, Association for Computing MachineryNew YorkNYUnited States

[10] Chicago Clojure 2017 - Stuart Halloway on Repl Driven Developmen

[11] Clojure in a nutshell by James Trunk

[12] https://youtu.be/dGVqrGmwOAw

[13] https://neanderthal.uncomplicate.org/articles/benchmarks.html

Sources III

[14] https://dragan.rocks/articles/20/Going-faster-than-Tensorflow-on-GPU-with-Clojure

[15] https://aiprobook.com/deep-learning-for-programmers/

[16] https://neanderthal.uncomplicate.org/articles/benchmarks.html

[17] https://mxnet.apache.org/

[18] https://djl.ai/

[19] http://gigasquidsoftware.com/blog/2018/12/18/how-to-gan-a-flan/

[20] http://www.genetic-programming.com/coursemainpage.html

[21] Conj 2017 - Carin Meier on "Deep Learning Needs Clojure"