Deep Learning in Julia

Deniz Yuret
Koç University, Istanbul
MLDAS-2019
machine learning in 5 slides
Machine learning: observations

Inputs

\[ x_1 \]
\[ x_2 \]
\[ \cdot \]
\[ \cdot \]
\[ x_n \]

Unknown process

Outputs

\[ y_1 \]
\[ y_2 \]
\[ \cdot \]
\[ \cdot \]
\[ y_n \]
Machine learning: modeling

Inputs $x_1, x_2, \ldots, x_n$

Differentiable program

Predictions $\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n$

$\text{ypred} = \text{predict}(w, x)$
Machine learning: loss function

Outputs

\[ y_1 \]
\[ y_2 \]
\[ \ldots \]
\[ y_n \]

Predictions

\[ \hat{y}_1 \]
\[ \hat{y}_2 \]
\[ \ldots \]
\[ \hat{y}_n \]

\[ \text{function} \quad \text{loss}(w, x, y) \]
\[ \quad \text{ypred} = \text{predict}(w, x) \]
\[ \quad \text{ydiff} = \text{ypred} - y \]
\[ \quad \text{sqerr} = \text{ydiff} .^2 \]
\[ \quad \text{qloss} = \text{sum}(\text{sqerr}) \]
\[ \text{end} \]
Machine Learning: optimization loop

function SGD(func, data)
    for args in data
        f = @diff func(args...)
        for w in params(f)
            w = w - grad(f,w) * learningRate
        end
    end
    return w
end
Deep learning software infrastructure

- Modeling language
  - Julia

- GPU support

- Automatic differentiation
  - Knet

- Optimization algorithms
Julia in 5 slides
Why Julia? (1/5) Speed

- Julia is fast.

- Most of Julia library is written in readable / hackable / composable Julia rather than opaque C++ (as opposed to Numpy, Tensorflow etc.)
Case study: Celeste project

- Classify stars and galaxies
- using Cori supercomputer at NERSC
- 188M objects from Sloan Digital Sky Survey
- 178TB of image data
- 14.6 minutes to load data, perform estimations
- Peak performance of 1.54 petaflops using 1.3 million threads on 9,300 nodes
- Written in pure Julia

https://juliacomputing.com/case-studies/celeste.html

Picture By ESA/Hubble, CC BY 4.0, https://commons.wikimedia.org/w/index.php?curid=68669699
Why Julia? (2/5) Multiple dispatch

- x::Matrix * y::Matrix
- x::Diagonal * y::Triangular
- x::KnetArray * y::KnetArray
- x::Param * y::Param
Why Julia? (3/5) Metaprogramming

- \( y = \tanh(w \times x + b) \)
- \( dy = \text{diff} \ \tanh(w \times x + b) \)
- \( \text{grad}(dy,w) \)
- \( \text{grad}(dy,b) \)
Why Julia? (4/5) Callable objects

```julia
struct Dense; w; b; f; end
(d::Dense)(x) = d.f.(d.w * x .+ d.b)

struct Chain; layers; end
(c::Chain)(x) = (for f in c.layers; x=f(x); end; x)
(c::Chain)(x,y) = nll(c(x),y)

d1 = Dense(randn(256,784),randn(256),relu)
d2 = Dense(randn(10,256),randn(10),identity)
mlp = Chain((d1,d2))
```
Why Julia? (5/5) Iterators

Lazy iterators for training and monitoring:

- Run one epoch: `sgd(f,d)`
- Run n epochs: `sgd(f,repeat(d,n))`
- Run n iters: `sgd(f,take(d,n))`
- Progress bar: `progress(sgd(f,d))`
- Run till conv: `converge(sgd(f,cycle(d)))`
- Do something every n iterations:
  `(task(x) for x in takenth(sgd(f,d),n))`
Evolution of computer languages
Evolution of deep learning frameworks

- Machine Code
- Assembler
- Fortran
- BASIC
- C
- CUDA
- ConvNet
- Caffe
- Torch
- Theano
- TFlow
- PyTorch
- DyNet
- TF Fold
- Julia
To explore further...

juliabox.com
github.com/denizyuret/Knet.jl