OCaml Workshop: Oxford, UK, September 8th, 2017.

# Owl - A General-Purpose Numerical Library in OCaml

Liang Wang

University of Cambridge

September 2017

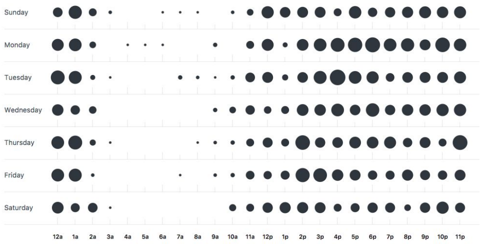
# Motivation

- How can we attract more people into OCaml community?
- OCaml core library is kept minimal (on purpose) for ?
- But many applications need strong numerical supports.
- Libraries for numerical computing in OCaml are fragmented.
- Difference between system library and numerical library:
    - A reduced set of operations atop of system abstraction.
    - A complex set of functions atop of number abstraction.

UNIVERSITY OF CAMBRIDGE

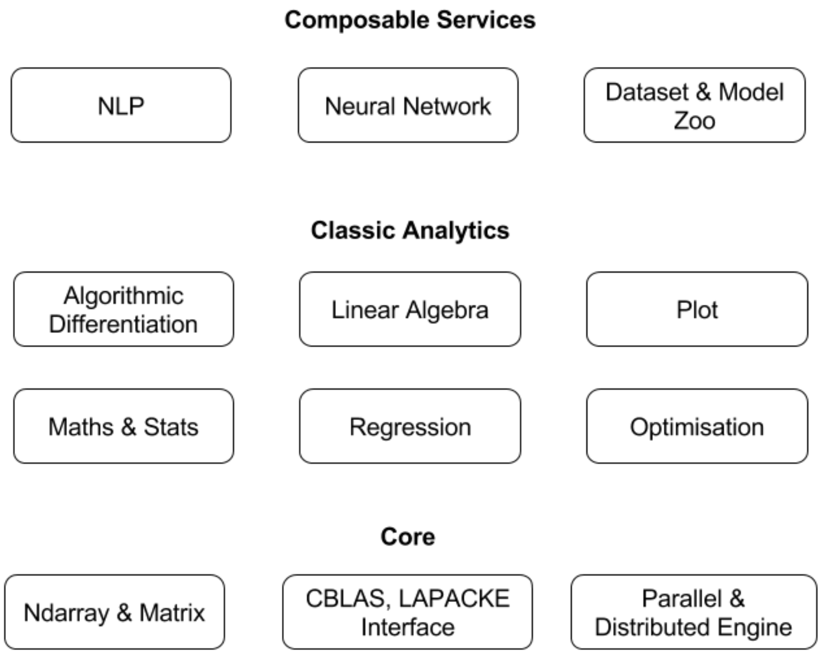
# Highlights

- Owl is an experimental project to develop a numerical platform.
- One year intensive development: > 130k LOC, 6.5k functions.
- A comprehensive set of classic numerical functions.
- Strong support for modern data analytics (ML & DNN).
- As fast as C but as concise as Python with static type-checking.


UNIVERSITY OF CAMBRIDGE

# Architecture




- Composable Services layer enables fast development of modern data analytics using type-safe functional programming.
- Classic Analytics layer implements a comprehensive set of classic analytic functions such as various hypothesis tests.
- Core layer provides the building block of the whole numerical system and the capability of distributed computing.

# Ndarray - Module Structure

- `Dense` and `Sparse` as two top modules.
- `S/D/C/Z` submodules to deal with four number types

```
Dense.Ndarray.S.zeros [|5;5|];;     (* single precision real ndarray *)
Dense.Ndarray.D.zeros [|5;5|];;     (* double precision real ndarray *)
Dense.Ndarray.C.zeros [|5;5|];;     (* single precision complex ndarray *)
Dense.Ndarray.Z.zeros [|5;5|];;     (* double precision complex ndarray *)
```

- There is also an `Any` module to deal with all other types.

```
let x = Dense.Ndarray.Any.create [|5;5;5|]  true in
  Dense.Ndarray.Any.get_slice_simple [[-1;0];[];[]] x;;
```

UNIVERSITY OF CAMBRIDGE

# Ndarray - Groups of Functions

- Creation functions: `zeros, gaussian, linspace, bernoulli, magic` ...
- Manipulation functions: `pad, slice, flip, rotate, transpose, tile` ...
- Iteration functions: `iter, map, filter, fold, map2, exists, for_all` ...
- Comparison functions: `equal, greater, less, elt_*, *_scalar` ...
- Vectorised uni- and binary maths functions: `sin, tanh, log, fix, elu ...`

UNIVERSITY OF CAMBRIDGE

# Ndarray - Performance Critical Code

```
#define FUN1 real_float_is_negative
#define NUMBER float
#define STOPFN(X) (X >= 0)
#include "owl_dense_common_vec_cmp.c"

#define FUN1 real_double_is_negative
#define NUMBER double
#define STOPFN(X) (X >= 0)
#include "owl_dense_common_vec_cmp.c"

#define FUN1 complex_float_is_negative
#define NUMBER complex_float
#define STOPFN(X) (X.r >= 0 || X.i >= 0)
#include "owl_dense_common_vec_cmp.c"

#define FUN1 complex_double_is_negative
#define NUMBER complex_double
#define STOPFN(X) (X.r >= 0 || X.i >= 0)
#include "owl_dense_common_vec_cmp.c"
```

```
#define FUN4 real_float_abs
#define NUMBER float
#define NUMBER1 float
#define MAPFN(X) (fabsf(X))
#include "owl_dense_common_vec_map.c"

....

#define FUN5 real_float_sum
#define INIT float r = 0.
#define NUMBER float
#define ACCFN(A,X) (A += X)
#define COPYNUM(X) (caml_copy_double(X))
#include "owl_dense_common_vec_fold.c"
```

# Ndarray - Indexing & Slicing

The most fundamental operation; the key to concise code.

- In Numpy : `x[ 1, 0:5; -1:0:-2 ]`
- In Julia : `x[ 1, 0:5, 1:end ]`
- In Owl : `get_slice_simple [ [1]; [0;5]; [-1;0;-2] ] x`

UNIVERSITY OF CAMBRIDGE

# Ndarray - More Complex Slicing

```
type index =
  | I of int          (* single index *)
  | L of int list     (* list of indices *)
  | R of int list     (* index range *)

type slice = index list

val get_slice : index list -> ('a, 'b) t -> ('a, 'b) t
val set_slice : index list -> ('a, 'b) t -> ('a, 'b) t -> unit
```

E.g., `[ I 5; L [2;1;0]; R [2;-1;2] ]`

# Ndarray - Operator Design

Commonly used operators have been implemented: `+ - * / +$ -$ = =. =~ > >. >$ ...`

Operators are divided into groups. E.g., Ndarray does not support `*@`

Operators are implemented as functors to avoid maintaining multiple copies of definition.

We can easily change the definition for the whole system easily in one place.

```
module Make_Basic (M : BasicSig) = struct
  type ('a, 'b) op_t0 = ('a, 'b) M.t
  let ( + ) = M.add
  let ( - ) = M.sub
  let ( * ) = M.mul
  let ( / ) = M.div
  let ( +$ ) = M.add_scalar
  ...
end
```

```
module Operator = struct
  include Owl_operator.Make_Basic (Owl_dense_matrix_generic)
  include Owl_operator.Make_Extend (Owl_dense_matrix_generic)
  include Owl_operator.Make_Matrix (Owl_dense_matrix_generic)
end
```

UNIVERSITY OF CAMBRIDGE

# Ndarray - Challenges

- Interoperation on different numbers - `Ext` module.

```
Ext.(F 10. * Dense.Matrix.C.uniform 5 5 >. F 5.)
```

- How about a generic abs using current Bigarray types?

```
external owl_real_float_min_i : int -> ('a, 'b) owl_vec -> int = "real_float_min_i"
external owl_real_double_min_i : int -> ('a, 'b) owl_vec -> int = "real_double_min_i"
external owl_complex_float_min_i : int -> ('a, 'b) owl_vec -> int = "complex_float_min_i"
external owl_complex_double_min_i : int -> ('a, 'b) owl_vec -> int =
"complex_double_min_i"

let _owl_min_i : type a b. (a, b) kind -> (a, b) owl_vec_op01 = function
  | Float32    -> owl_real_float_min_i
  | Float64    -> owl_real_double_min_i
  | Complex32 -> owl_complex_float_min_i
  | Complex64 -> owl_complex_double_min_i
  | _          -> failwith "_owl_min_i: unsupported operation"
```

UNIVERSITY OF CAMBRIDGE

# CBLAS/LAPACKE Interface

- Not BLAS/LAPACK because Owl sticks to C-layout.
- Interface is automatically generated from C header files.
- Rely on Ctypes, but only used its c stub file (`foreign` is an issue).
- The challenge is the versioning and deployment on different platforms.

UNIVERSITY OF CAMBRIDGE

# CBLAS/LAPACKE Interface to Linalg

- Automatically generated code only provides a thin wrapping.
- Still need hand-writing interface to wrap up `S/D/C/Z` types.
- How much to expose to Linalge? Flexibility vs. Convenience.
- It is often believed that C-interface introduces some overhead, but Owl often achieves better performance than Julia.

UNIVERSITY OF CAMBRIDGE

# Distributed Computing - Design Rationale

Think about distributed analytics -> it is distribution + analytics

- Parallel & distributed engine must be separated out.
- Engine APIs must be minimal and simple for easy composition.
- Able to deal with both low-level and high-level data structures.
- Avoid letting developers deal with details like message passing.

UNIVERSITY OF CAMBRIDGE

# Owl + Actor (Sub)System

- Three engines: Map-Reduce; Parameter Server; Peer-to-Peer
- Specifically designed synchronisation mechanism for scalability.

```
val start : ?barrier:barrier -> string -> string -> unit
val register_barrier : ps_barrier_typ -> unit
val register_schedule : ('a, 'b, 'c) ps_schedule_typ -> unit
val register_pull : ('a, 'b, 'c) ps_pull_typ -> unit
val register_push : ('a, 'b, 'c) ps_push_typ -> unit
val register_stop : ps_stop_typ -> unit
```

```
type barrier =
  | ASP    (* Asynchronous Parallel *)
  | BSP    (* Bulk Synchronous Parallel *)
  | SSP    (* Stale Synchronous Parallel *)
  | PSP    (* Probabilistic Synchronous Parallel *)
```

UNIVERSITY OF CAMBRIDGE

# Owl + Actor : Neural Network Example

```
let network =
  input [|28;28;1|]
  |> lambda (fun x -> Maths.(x / F 256.))
  |> conv2d [|5;5;1;32|] [|1;1|] ~act_typ:Activation.Relu
  |> max_pool2d [|2;2|] [|2;2|]
  |> dropout 0.1
  |> fully_connected 1024 ~act_typ:Activation.Relu
  |> linear 10 ~act_typ:Activation.Softmax
  |> get_network
```

A convolutional neural network can be defined as concise as that in the state-of-the-art system specialised in deep neural networks.

Changing it into a distributed algorithm just requires one line of code thanks to Owl's cutting edge parallel & distributed computation engine.

Make it distributed! Yay!

```
module M2 = Owl_neural_parallel.Make ( Owl.Neural.S.Graph )
                                      (Actor.Param)
```

# Owl + Actor : Ndarray Example

Similarly, we can also transform a Ndarray into a distributed Ndarray, by simply

```
module M2 = Owl_parallel.Make ( Dense.Ndarray.S) (Actor.Mapre)
```

Composed by a functor in `Owl_parallel` module, which connects two systems and hides details.

```
module type Mapre_Engine = sig
  val map : ('a -> 'b) -> string -> string
  val map_partition: ('a list -> 'b list) -> string -> string
  val union : string -> string -> string
  val reduce : ('a -> 'a -> 'a) -> string -> 'a option
  val collect : string -> 'a list
  val workers : unit -> string list
  val myself : unit -> string
  val load : string -> string
  val save : string -> string -> int
end
```

```
module type Ndarray = sig
  val shape : arr -> int array
  val empty : int array -> arr
  val create : int array -> elt -> arr
  val zeros : int array -> arr
  ...
end
```

UNIVERSITY OF CAMBRIDGE

# Algorithmic Differentiation

- Core component to bridge the gap between low-level numerical functions and high-level analytical models.
- Functor to support both single- and double-precision.
- Support quite many functions and operations already.

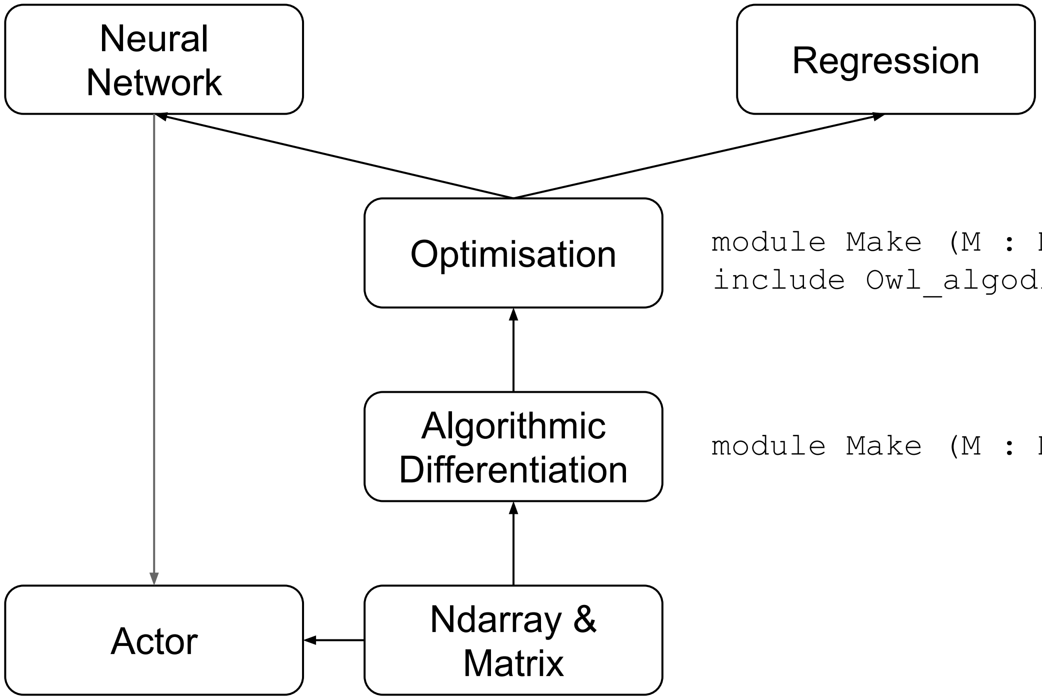
```
(* AD module of Float32 type *)
module S = Owl_algodiff_generic.Make (Owl_dense_matrix.S) (Owl_dense_ndarray.S)

(* AD module of Float64 type *)
module D = Owl_algodiff_generic.Make (Owl_dense_matrix.D) (Owl_dense_ndarray.D)
```

UNIVERSITY OF CAMBRIDGE

# Revisit Owl's Architecture

```
module Make (M : ...) (A : ...) = struct
include Owl_optimise_generic.Make (M) (A)
```

```
module Make (M : ...) (A : ...) = struct
include Owl_optimise_generic.Make (M) (A)
```




```
module Make (M : MatrixSig) (A : NdarraySig) = struct
include Owl_algodiff_generic.Make (M) (A)
```

```
module Make (M : MatrixSig) (A : NdarraySig)
```

UNIVERSITY OF CAMBRIDGE

# Regression - OLS

```
let ols ?(i=false) x y =
    let params = Params.config
      ~batch:(Batch.Full) ~learning_rate:(Learning_Rate.Adagrad 1.)
      ~gradient:(Gradient.GD) ~loss:(Loss.Quadratic) ~verbosity:false
      ~stopping:(Stopping.Const 1e-16) 1000.
    in
    _linear_reg i params x y
```

UNIVERSITY OF CAMBRIDGE

# Regression - Ridge

```
let ridge ?(i=false) ?(a=0.001) x y =
    let params = Params.config
      ~batch:(Batch.Full) ~learning_rate:(Learning_Rate.Adagrad 1.)
      ~gradient:(Gradient.GD) ~loss:(Loss.Quadratic)
      ~regularisation:(Regularisation.L2norm a) ~verbosity:false
      ~stopping:(Stopping.Const 1e-16) 1000.
    in
    _linear_reg i params x y
```

# Regression - SVM

```
let svm ?(i=false) ?(a=0.001) x y =
    let params = Params.config
      ~batch:(Batch.Full) ~learning_rate:(Learning_Rate.Adagrad 1.)
      ~gradient:(Gradient.GD) ~loss:(Loss.Hinge)
      ~regularisation:(Regularisation.L2norm a) ~verbosity:true
      ~stopping:(Stopping.Const 1e-16) 1000.
    in
    _linear_reg i params x y
```

# Neural Network

- Built atop of `Algodiff` Module, pros > cons
- Support both simple Feedforward and Graph structure.
- Many neurons have been implemented: linear, conv2d, maxpool ...
- Code is as concise as state-of-the-art specialised library.
- Performance can be improved by Actor system.

# Neural Network - VGG Example

```
let make_network input_shape =
  input input_shape
  |> normalisation ~decay:0.9
  |> conv2d [|3;3;3;32|] [|1;1|] ~act_typ:Activation.Relu
  |> conv2d [|3;3;32;32|] [|1;1|] ~act_typ:Activation.Relu ~padding:VALID
  |> max_pool2d [|2;2|] [|2;2|] ~padding:VALID
  |> dropout 0.1
  |> conv2d [|3;3;32;64|] [|1;1|] ~act_typ:Activation.Relu
  |> conv2d [|3;3;64;64|] [|1;1|] ~act_typ:Activation.Relu ~padding:VALID
  |> max_pool2d [|2;2|] [|2;2|] ~padding:VALID
  |> dropout 0.1
  |> fully_connected 512 ~act_typ:Activation.Relu
  |> linear 10 ~act_typ:Activation.Softmax
  |> get_network
```

Define a network structure in a very concise and functional way.

UNIVERSITY OF CAMBRIDGE

# Neural Network - Cost of ...

```
module Normalisation = struct

  let run x l =
    let a = F (1. /. float_of_int (shape x).(l.axis)) in
    l.mu <- Maths.(a * (sum_ ~axis:l.axis x));
    l.var <- Maths.(a * (sum_ ~axis:l.axis (x * x)));
    let x' = Maths.((x - l.mu) / sqrt (l.var + F 1e-8)) in
    Maths.(x' * l.gamma + l.beta)

  ...

end
```

**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: $\gamma$, $\beta$

**Output:** $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

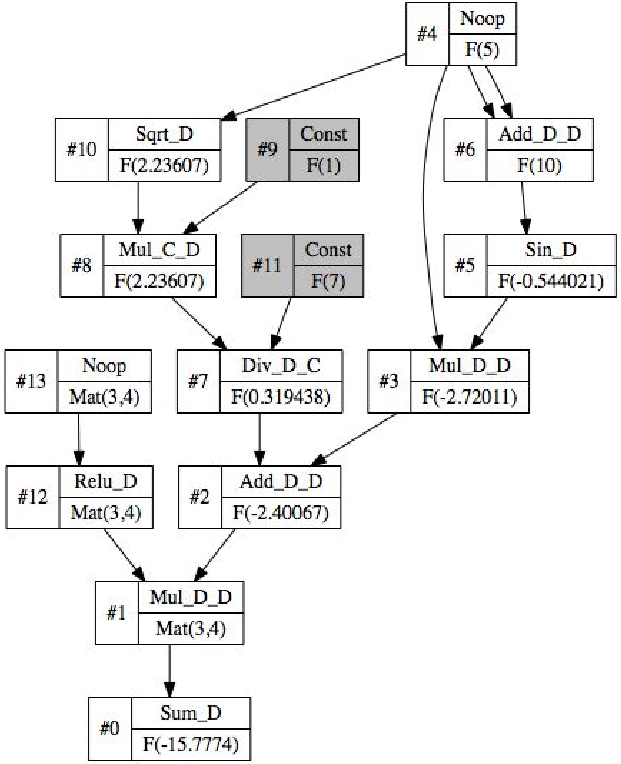
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Adding a new neuron type is simple, `Algodiff` will take care of calculating derivatives in the backpropagation phase.

UNIVERSITY OF CAMBRIDGE

# Computation Graph - Simple Function

```
let f x y = Maths.((x * sin (x + x) + ( F 1. * sqrt x) / F 7.) * (relu y) |> sum)
```



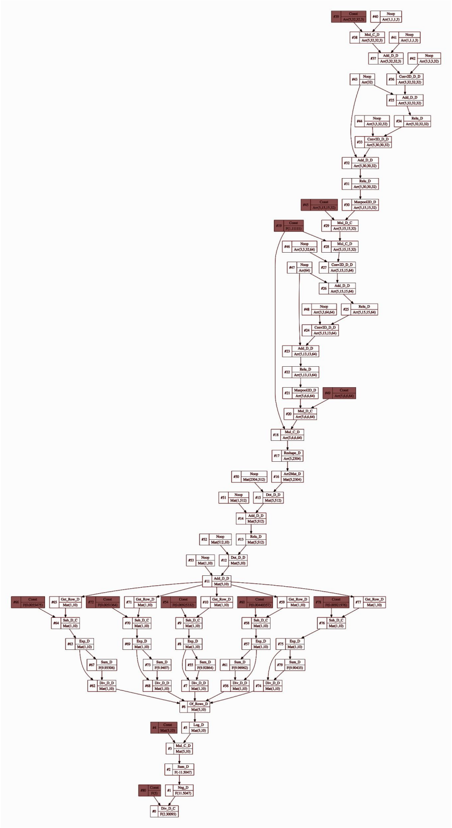

Debugging can be tricky, and visualisation is often very helpful.

Owl can print out raw trace on terminal in human-readable format, or print in dot file format for further visualisation.

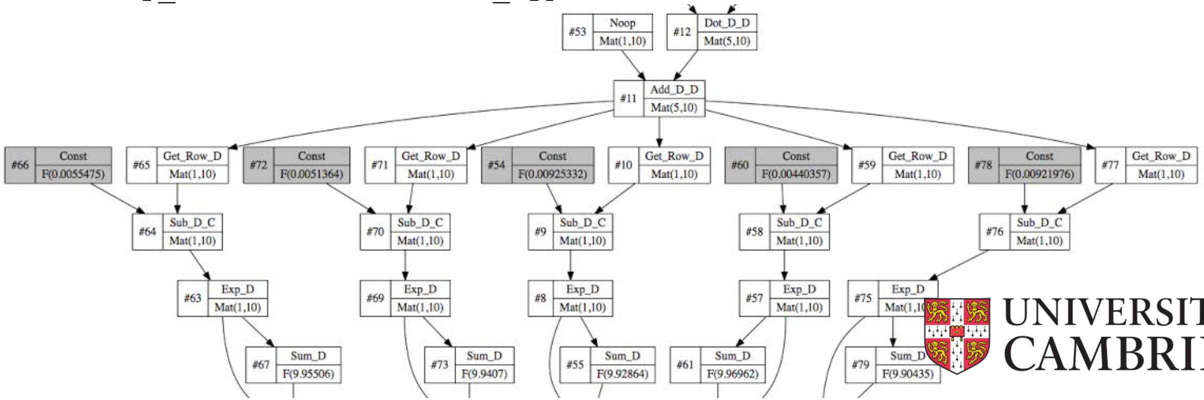
The node contains rich information for debugging.

UNIVERSITY OF CAMBRIDGE

# Computation Graph - VGG Network

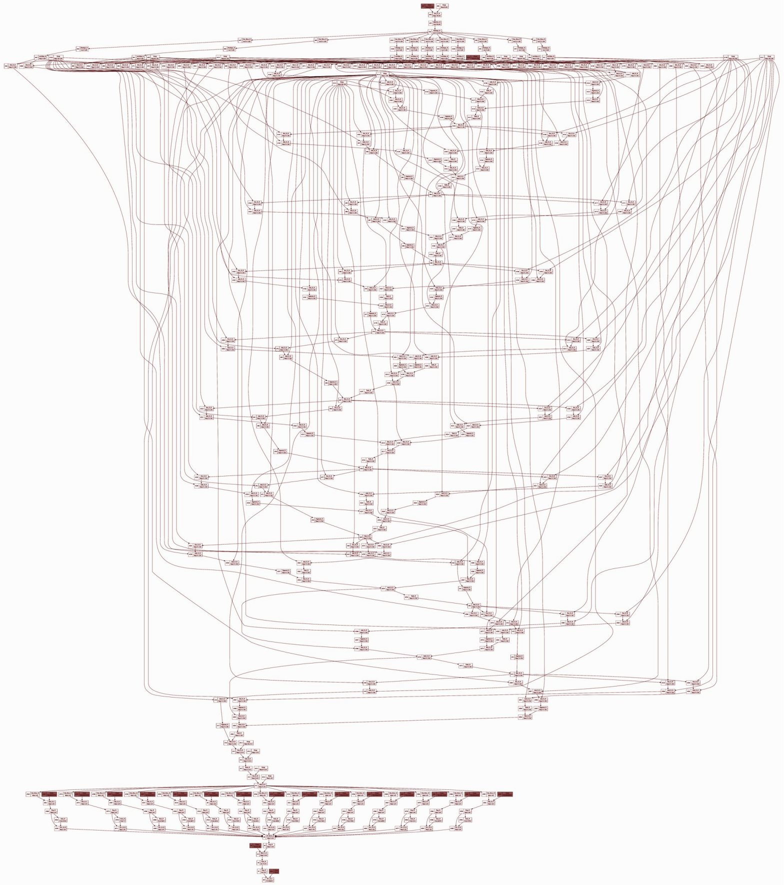


```
input input_shape
|> normalisation
|> conv2d [|3;3;3;32|] [|1;1|] ~act_typ:Activation.Relu
|> conv2d [|3;3;32;32|] [|1;1|] ~act_typ:Activation.Relu
~padding:VALID
|> max_pool2d [|2;2|] [|2;2|] ~padding:VALID
|> dropout 0.1
|> conv2d [|3;3;32;64|] [|1;1|] ~act_typ:Activation.Relu
|> conv2d [|3;3;64;64|] [|1;1|] ~act_typ:Activation.Relu
~padding:VALID
|> max_pool2d [|2;2|] [|2;2|] ~padding:VALID
|> dropout 0.1
|> fully_connected 512 ~act_typ:Activation.Relu
```




UNIVERSITY OF CAMBRIDGE

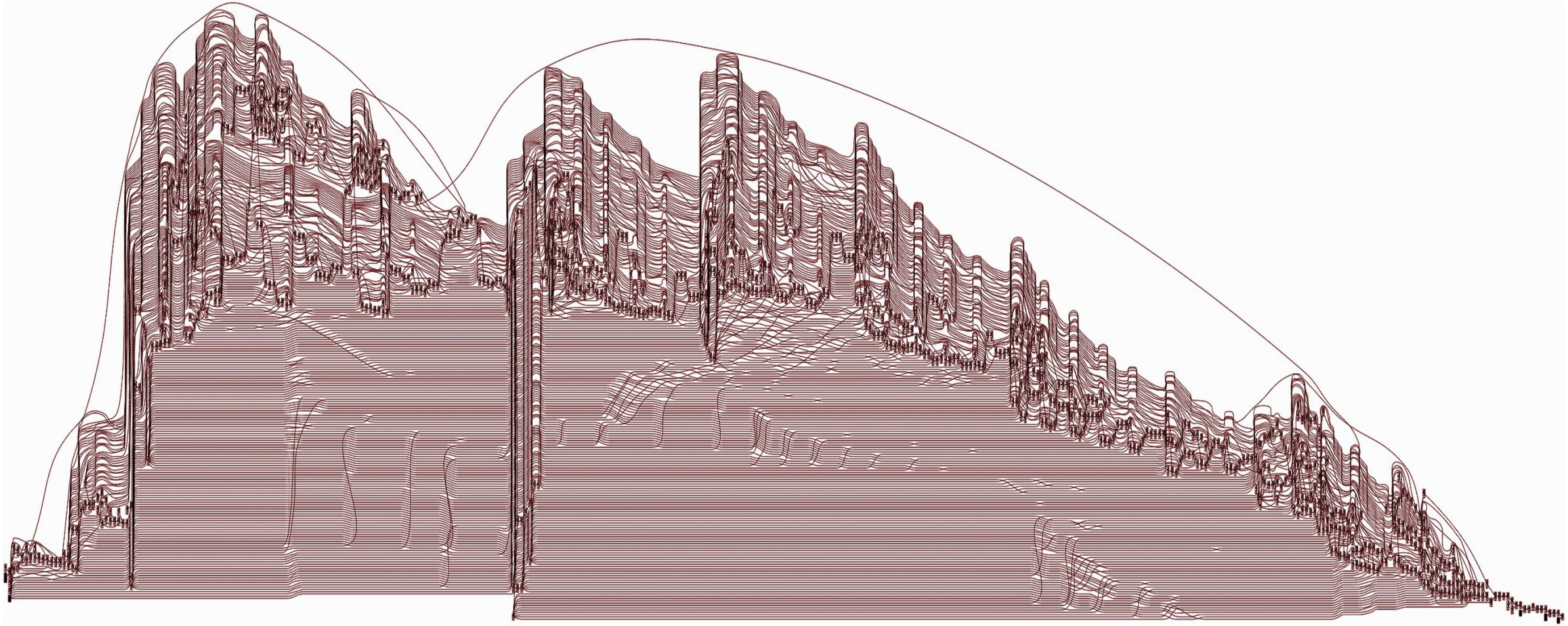
# Computation Graph - LSTM Network



```
input [|wndsz|]
|> embedding vocabsz 40
|> lstm 128
|> linear 512 ~act_typ:Activation.Relu
|> linear vocabsz ~act_typ:Activation.Softmax
|> get_network
```

UNIVERSITY OF CAMBRIDGE

# Google Inception-V3

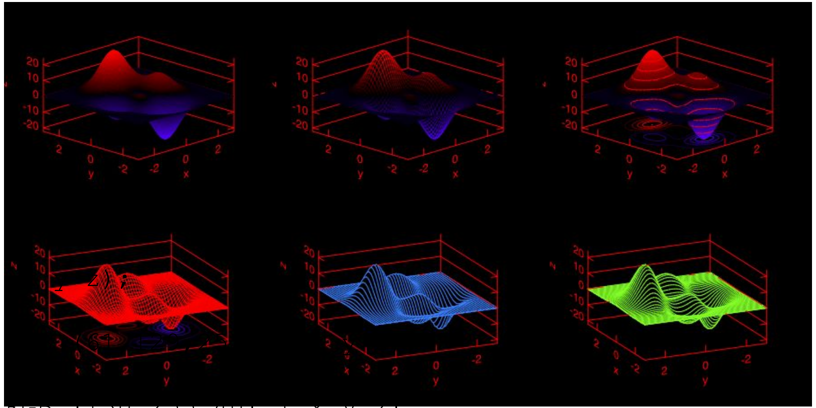


UNIVERSITY OF CAMBRIDGE

# Plotting Functions

- Built atop of `ocaml-plplot`, but with improved interface.
- Hide all the dirty details in Plplot, provide matlab-like APIs.
- The core plotting engine is very small < 200 LOC

```
let h = Plot.create ~m:2 ~n:3 "plot_017.png" in
Plot.subplot h 0 0;
Plot.surf ~h x y z;
Plot.subplot h 0 1;
Plot.mesh ~h x y z;
Plot.subplot h 0 2;
Plot.(surf ~h ~spec:[ Contour ] x y z);
Plot.subplot h 1 0;
Plot.(mesh ~h ~spec:[ Contour; Azimuth 115.; NoMagColor ] x y z);
Plot.subplot h 1 1;
Plot.(mesh ~h ~spec:[ Azimuth 115.; ZLine X; NoMagColor;
Plot.subplot h 1 2;
Plot.(mesh ~h ~spec:[ Azimuth 115.; ZLine Y; NoMagColor; RGB (150,255,40) ] x y z);
Plot.output h;;
```



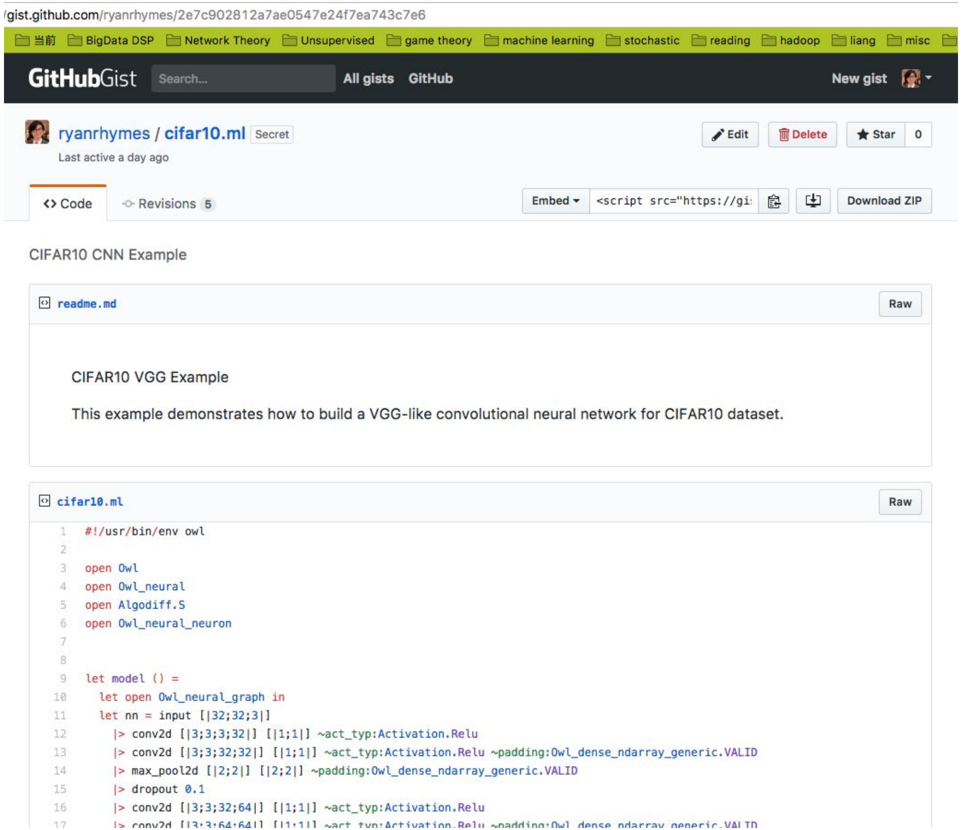
UNIVERSITY OF CAMBRIDGE

# Zoo System - Share Code Snippets

- Originally designed for sharing neural network models.
- Now used for sharing small code snippet and scripting in Owl.
- Enhance Owl's numerical capability and fast prototyping.
- Very simple design: add one more directive `#zoo` to toplevel.

UNIVERSITY OF CAMBRIDGE

# Zoo System - Example




Publish on gist; then use in your script with `#zoo` directive.

```
#!/usr/bin/env owl

#zoo "2e7c902812a7ae0547e24f7ea743c7e6"
#zoo "217ef87bc36845c4e78e398d52bc4c5b"
```

```
Owl's Zoo System
Usage:
  owl [utop options] [script-file]
  owl -upload [gist-directory]
  owl -download [gist-id]
  owl -remove [gist-id]
  owl -update [gist-ids]
  owl -run [gist-id]
  owl -info [gist-ids]
  owl -list
  owl -help
```

UNIVERSITY OF CAMBRIDGE

# Future Plan

- There is a great space for further optimisation.
- C code can be further optimised with SSE …
- Actor will include an engine for multicore OCaml.
- Matrix and Ndarray can be unified, leads to simpler code.
- Indexing needs enhancement, ppx, performance, and etc.

UNIVERSITY OF CAMBRIDGE

Thank you!

Website: https://github.com/ryanrhymes/owl

UNIVERSITY OF CAMBRIDGE

# ....

- How to architect a modern numerical library
- Meet modern need.
- Extensible, Flexible, Manageable
- Focus on architectural design, avoid doing thing that can be done by the machine: generate interface, calculating derivative
- Do as much as we can with as little code as possible.
- Why … reduce potential error.

UNIVERSITY OF CAMBRIDGE