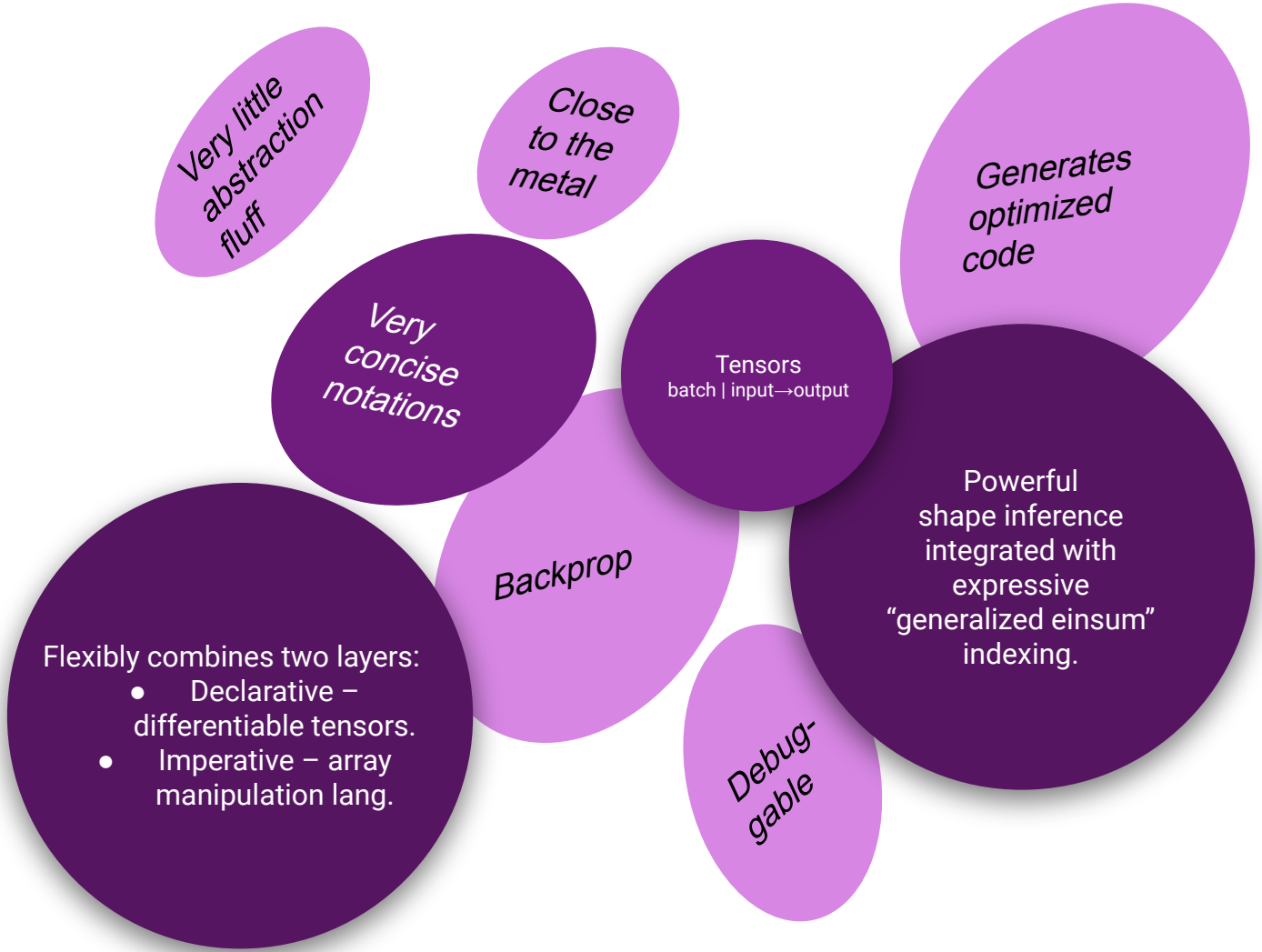
OCANNL

Mysteries of NN training unveiled

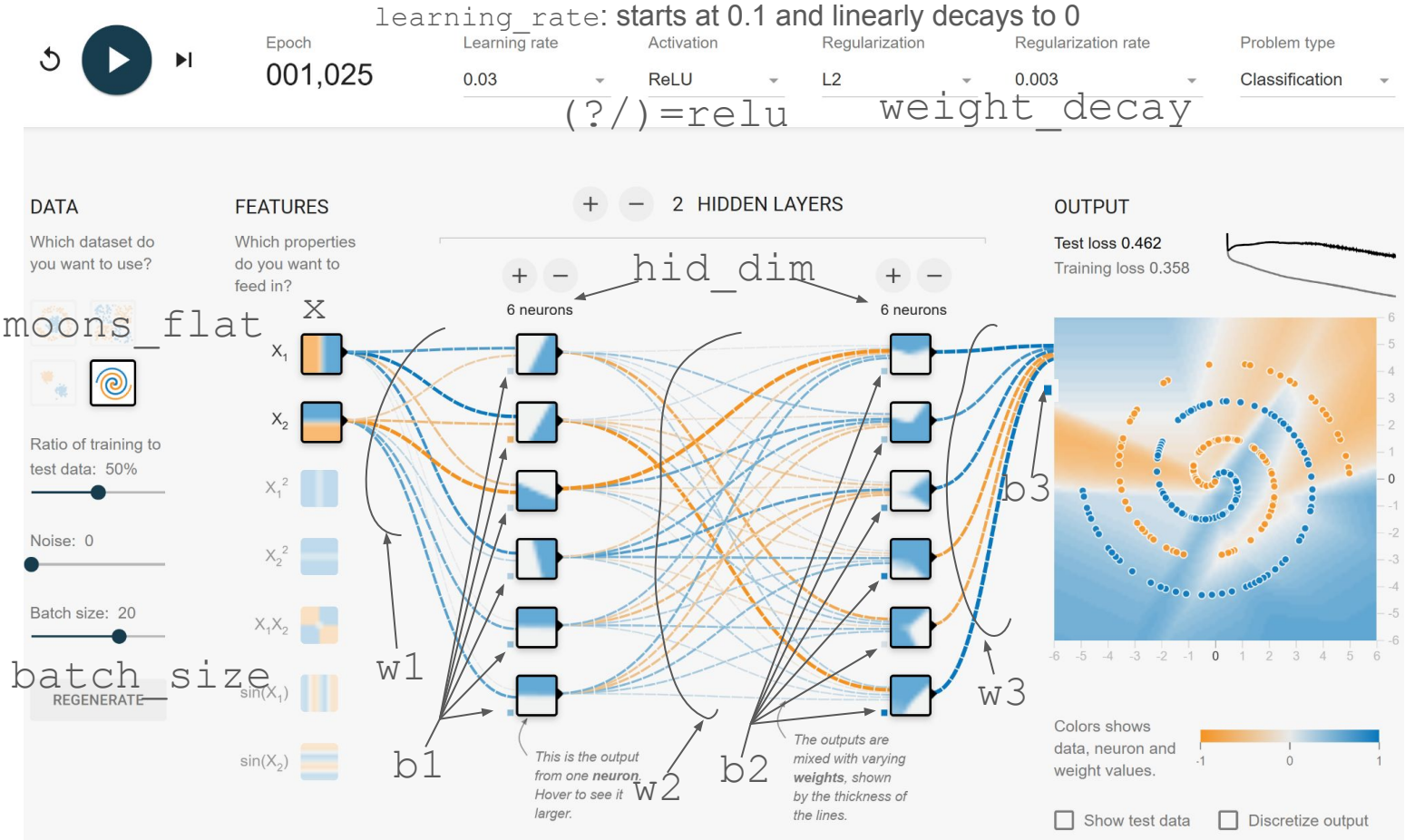
OCaml **C**ompiles **A**lgorithms for **N**eural **N**etwork **L**earning

- OCANNL is distributed as two opam packages:
  - `arrayjit`: a backend for compilers for numerical array programming languages,
  - **`n`**`eural_`**`n`**`ets_`**`l`**`ib`: a neural networks (Deep Learning) framework.
- You express numerical computations at runtime, OCANNL will optimize them for the runtime-dependent shape of your data, compile and dynamically load them using one of its backends.
- There are startups doing it in other languages, so it must be worth it!
  - In Python: tinygrad – democratizing DL – at home or on premise training of large models.
  - In Rust: Luminal – simplifies deployment of large models with on-device inference.
  - OCaml is a good fit for writing optimizing compilers.
- Value added:
  - OCANNL has concise notation thanks to better shape inference (i.e. type inference for the data and transformation matrices).
  - OCANNL is explicit about the backends (and devices) used, which might make it a good fit for heterogeneous computing (e.g. combining GPUs from different companies).

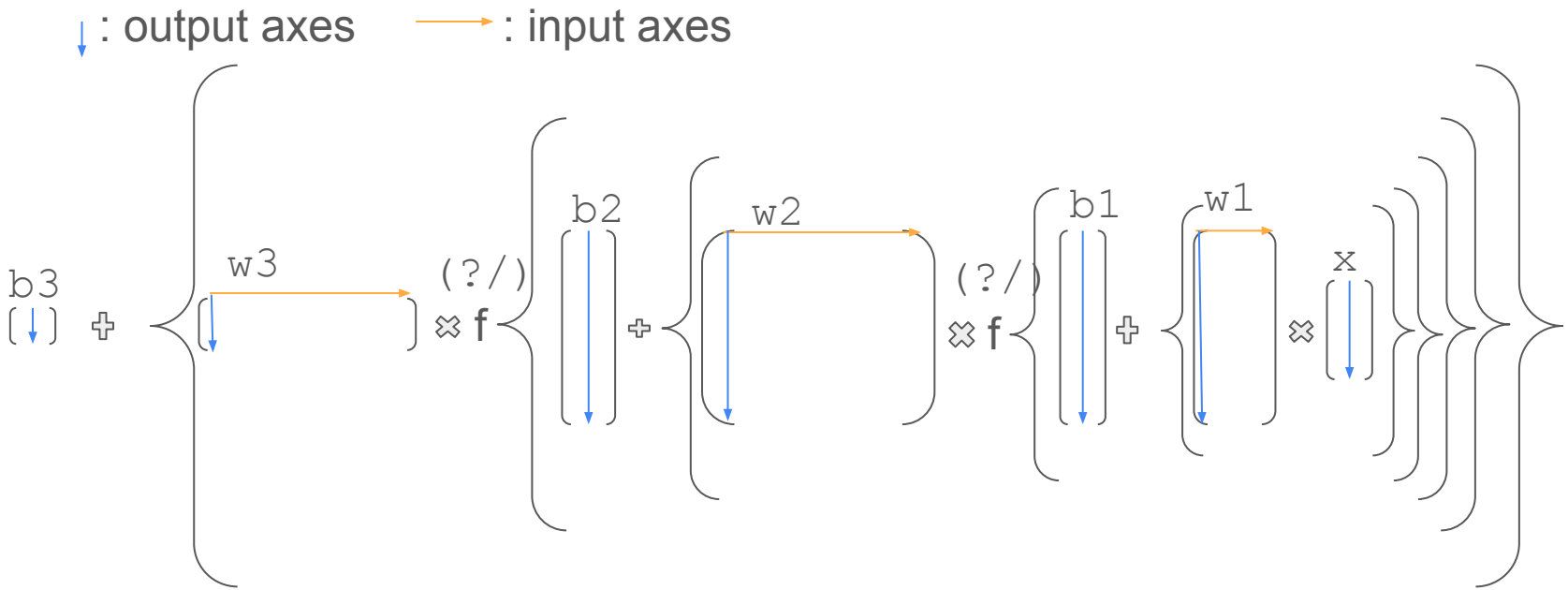

Very little abstraction fluff
Close to the metal
Generates optimized code
Very concise notations
Tensors
batch | input→output
Powerful shape inference integrated with expressive “generalized einsum” indexing.
Backprop
Flexibly combines two layers:
• Declarative – differentiable tensors.
• Imperative – array manipulation lang.
Debug-gable

Let’s train a feed-forward neural network with 2 hidden layers (aka. a 3-layer MLP) to classify points on a plane.

[Try Tensorflow Playground for yourself.](#) [Compare with bin/moons_demo.ml:](#)



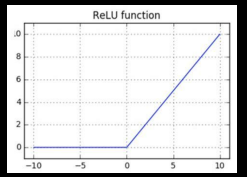
Tensors as differentiable multidimensional matrices



Multi Layer Perceptron in one line of code

```
let%op mlp x = "b3" + ("w3" * ?/("b2" hid_dim + ("w2" * ?/("b1" hid_dim + ("w1" * x))))) in
```

?/ =



Tensor (e.g. matrix) multiplication

Introduces identifier `w1` for a parameter tensor

Sets the output dimension of `b1` to `hid_dim`

“Rectified Linear Unit” unary operation, see graph on top-right

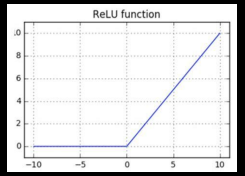
Tensor function, expands to:
`let w1 = … in let b1 = … in … let mlp x = … in …`

Declarative expressions for differentiable tensor operations

Multi Layer Perceptron in one line of code

```
let%op mlp x = "b3" + ("w3" * ?/("b2" hid_dim + ("w2" * ?/("b1" hid_dim + ("w1" * x))))) in
```

?/ =




```
let moons_flat =
  Array.concat_map (Array.create ~len ())
    ~f:
      Float.(
        fun () ->
          let i = Rand.int len in
          let v = of_int i * pi / of_int len in
          let c = cos v and s = sin v in
          [| c + noise (); s + noise (); 1.0 - c + noise (); 0.5 - s + noise () |])
in
let moons_flat = TDSL.init_const ~l:"moons_flat" ~o:[ 2 ] moons_flat in
let moons_classes = Array.init (len * 2) ~f:(fun i -> if i % 2 = 0 then 1. else -1.) in
let moons_classes = TDSL.init_const ~l:"moons_classes" ~o:[ 1 ] moons_classes in
let batch_n, bindings = IDX.get_static_symbol ~static_range:n_batches IDX.empty in
let step_n, bindings = IDX.get_static_symbol bindings in
let%op moons_input = moons_flat @| batch_n in
let%op moons_class = moons_classes @| batch_n in
let%op margin_loss = ?/(1 - (moons_class *. mlp moons_input)) in
let%op scalar_loss = (margin_loss ++ "...|... => 0") /. !..batch_size in
let update = Train.grad_update scalar_loss in
let%op learning_rate = 0.1 *. (!..steps - !@step_n) /. !..steps in
Train.set_hosted learning_rate.value;
let sgd = Train.sgd_update ~learning_rate ~weight_decay update in
```

Specify just the output dimensions, batch dimensions will be inferred

`*.` is pointwise multiplication

Indices set in OCaml and passed to the backend code

Slice the tensors at their last batch axis

Pointwise division by the integer `steps`

Sum out all batch and output axes i.e. sum all values of `margin_loss`

Vanilla training loop

```
let module Backend = (val Train.fresh_backend ()) in
let device = Backend.(new_virtual_device @@ get_device ~ordinal:0) in
let ctx = Backend.init device in
let routine = Backend.(link ctx @@ compile bindings (Seq (update.fwd_bprop, sgd))) in
Train.all_host_to_device (module Backend) routine.context scalar_loss;
Train.all_host_to_device (module Backend) routine.context learning_rate;
let open Operation.At in
let step_ref = IDX.find_exn routine.bindings step_n in
let batch_ref = IDX.find_exn routine.bindings batch_n in
step_ref := 0;
let%track_this_sexp _train_loop : unit =
  for epoch = 0 to epochs - 1 do
    for batch = 0 to n_batches - 1 do
      batch_ref := batch;
      Train.run routine;
      assert (Backend.to_host routine.context learning_rate.value);
      assert (Backend.to_host routine.context scalar_loss.value);
      Backend.await device;
      learning_rates := learning_rate.@[0] :: !learning_rates;
      losses := scalar_loss.@[0] :: !losses;
      epoch_loss := !epoch_loss +. scalar_loss.@[0];
      Int.incr step_ref
    done;
    Stdio.printf "Epoch %d, lr=%f, epoch loss=%f\n%!" epoch learning_rate.@[0] !epoch_loss;
    epoch_loss := 0.
  done
in
```

This computes values and gradients for nodes of the tensor expression `scalar_loss`

This computes the update of the parameters inside the `scalar_loss` expression

That's for ppx_minidebug tracing

Tell the device to run the code when it's ready, and then to copy learning_rate and scalar_loss for accessing from OCaml.

Almost: `scalar_loss.value.array.{[|0|]}`

Vanilla inference loop

```
let points = Tensor.value_2d_points ~xdim:0 ~ydim:1 moons_flat in
let classes = Tensor.value_1d_points ~xdim:0 moons_classes in
let points1, points2 = Array.partitioni_tf points ~f:Float.(fun i _ -> classes.(i) > 0.) in
let%op mlp_result = mlp "point" in
Train.set_on_host Changed_on_devices mlp_result.value;
let result_routine =
  Backend.(
    link routine.context @@ compile IDX.empty @@ Block_comment ("moons infer", mlp_result.forward))
in
let callback (x, y) =
  Tensor.set_values point [| x; y |];
  assert (Backend.from_host result_routine.context point.value);
  Train.run result_routine;
  assert (Backend.to_host result_routine.context mlp_result.value);
  Backend.await device;
  Float.(mlp_result.@[0] >= 0.)
in
```

We want it in OCaml, don't optimize it away

Tell the device to fetch `point.value` from the OCaml side

```
let%track_this_sexp _plotting : unit =
  let plot_moons =
    let open PrintBox_utils in
    plot ~size:(120, 40) ~x_label:"ixes" ~y_label:"ygreks"
      [
        Scatterplot { points = points1; pixel = "#" };
        Scatterplot { points = points2; pixel = "%" };
        Boundary_map { pixel_false = "."; pixel_true = "*"; callback };
      ]
  in
  Stdio.printf "\nHalf-moons scatterplot and decision boundary:\n%!";
  PrintBox_text.output Stdio.stdout plot_moons
in
Backend.unsafe_cleanup ~unsafe_shutdown:true ()
```

Training run and result

```
lukstafi@DESKTOP-6RRUNR4:~/ocannl$ dune exec bin/moons_demo.exe

Welcome to OCANNL! Reading configuration defaults from /home/lukstafi/ocannl
Retrieving commandline, environment, or config file variable ocannl_backend
Not found, using default cc
```

```
Epoch 71, lr=0.004004, epoch loss=0.073218
Epoch 72, lr=0.002684, epoch loss=0.071019
Epoch 73, lr=0.001364, epoch loss=0.070818
Epoch 74, lr=0.000044, epoch loss=0.069533

Half-moons scatterplot and decision boundary:
 1.097e+0
y
g
r
e
k
s
-5.724e-1
-1.056e+0                                                    2.089e+0
```

Deep dive: Stochastic Gradient Descent

SGD = Stochastic Gradient Descent

SGD, or SGD w/ momentum, or Nesterov, in ~6 lines of code

```
let sgd_one ~learning_rate ?(momentum = 0.0) ?(weight_decay = 0.0) ?(nesterov = false) p =
  let sgd_delta = NTDSL.term ~label:("sgd_delta" :: p.Tensor.value.label) () in
  let sgd_momentum = NTDSL.term ~label:("sgd_momentum" :: p.value.label) () in
  [%cd
    ~~(p "param sgd step");
    sgd_delta =: p.grad + (!.weight_decay *. p);
    if Float.(momentum > 0.0) then (
      sgd_momentum =: (!.momentum *. sgd_momentum) + sgd_delta;
      if nesterov then sgd_delta =+ !.momentum *. sgd_momentum else sgd_delta =: sgd_momentum);
    p =- learning_rate *. sgd_delta]

let sgd_update ~learning_rate ?momentum ?weight_decay ?nesterov l =
  let code =
    l.params |> Set.to_list
    |> List.map ~f:(sgd_one ~learning_rate ?momentum ?weight_decay ?nesterov)
    |> Asgns.sequential
  in
  Asgns.Block_comment (l.label ^ " sgd update", code)
```

~~ means comment extending to end of scope
=: means assignment, `b =+ c` is `b =: b+c`

SGD update step: Assignments representation

```
# "scalar_loss sgd update";
# "b1 param sgd step";
n60 := 0.0002;
n61 =: n60 * b1 ~logic:".";
sgd_delta_b1 =: b1.grad + n61;
n50 := 2250;
n51 := !@i2;
n52 := 2250;
n53 =: n52 - n51;
n54 := 0.1;
n55 =: n54 * n53 ~logic:".";
learning_rate =: n55 / n50 ~logic:".";
n59 =: learning_rate * sgd_delta_b1 ~logic:".";
b1 =- n59;
# "b2 param sgd step";
n65 := 0.0002;
n66 =: n65 * b2 ~logic:".";
sgd_delta_b2 =: b2.grad + n66;
n64 =: learning_rate * sgd_delta_b2 ~logic:".";
b2 =- n64;
```

perform pointwise

:= is initialization or data fetching

!@ converts int to tensor

```
# "b3 param sgd step";
n70 := 0.0002;
n71 =: n70 * b3 ~logic:".";
sgd_delta_b3 =: b3.grad + n71;
n69 =: learning_rate * sgd_delta_b3 ~logic:".";
b3 =- n69;
# "w1 param sgd step";
n75 := 0.0002;
n76 =: n75 * w1 ~logic:".";
sgd_delta_w1 =: w1.grad + n76;
n74 =: learning_rate * sgd_delta_w1 ~logic:".";
w1 =- n74;
# "w2 param sgd step";
n80 := 0.0002;
n81 =: n80 * w2 ~logic:".";
sgd_delta_w2 =: w2.grad + n81;
n79 =: learning_rate * sgd_delta_w2 ~logic:".";
w2 =- n79;
# "w3 param sgd step";
n85 := 0.0002;
n86 =: n85 * w3 ~logic:".";
sgd_delta_w3 =: w3.grad + n86;
n84 =: learning_rate * sgd_delta_w3 ~logic:".";
w3 =- n84;
```

SGD update step: Lowered representation

```
/* scalar_loss sgd update */
/* b1 param sgd step */
n60[0] := 0.000200;
for i84 = 0 to 15 {
  n61[i84] := (n60[0] * b1[i84]);
}
for i86 = 0 to 15 {
  sgd_delta_b1[i86] := (b1.grad[i86] + n61[i86]);
}
n50[0] := 2250.000000;
n51[0] := i2;
n52[0] := 2250.000000;
n53[0] := (n52[0] - n51[0]);
n54[0] := 0.100000;
n55[0] := (n54[0] * n53[0]);
learning_rate[0] := (n55[0] / n50[0]);
for i88 = 0 to 15 {
  n59[i88] := (learning_rate[0] * sgd_delta_b1[i88]);
}
for i90 = 0 to 15 {
  b1[i90] := (b1[i90] - n59[i90]);
}
/* end */
/* b2 param sgd step */
n65[0] := 0.000200;
for i92 = 0 to 15 {
  n66[i92] := (n65[0] * b2[i92]);
}
for i94 = 0 to 15 {
  sgd_delta_b2[i94] := (b2.grad[i94] + n66[i94]);
}

for i96 = 0 to 15 {
  n64[i96] := (learning_rate[0] * sgd_delta_b2[i96]);
}
for i98 = 0 to 15 {
  b2[i98] := (b2[i98] - n64[i98]);
}
/* end */
/* b3 param sgd step */
n70[0] := 0.000200;
n71[0] := (n70[0] * b3[0]);
sgd_delta_b3[0] := (b3.grad[0] + n71[0]);

n69[0] := (learning_rate[0] * sgd_delta_b3[0]);
b3[0] := (b3[0] - n69[0]);
/* end */
```

```
/* w1 param sgd step */
n75[0] := 0.000200;
for i101 = 0 to 15 {
  for i102 = 0 to 1 {
    n76[i101, i102] := (n75[0] * w1[i101, i102]);
  }
}
for i105 = 0 to 15 {
  for i106 = 0 to 1 {
    sgd_delta_w1[i105, i106] := (w1.grad[i105, i106] + n76[i105, i106]);
  }
}

for i109 = 0 to 15 {
  for i110 = 0 to 1 {
    n74[i109, i110] := (learning_rate[0] * sgd_delta_w1[i109, i110]);
  }
}
for i113 = 0 to 15 {
  for i114 = 0 to 1 {
    w1[i113, i114] := (w1[i113, i114] - n74[i113, i114]);
  }
}
/* end */
/* w2 param sgd step */
n80[0] := 0.000200;
for i117 = 0 to 15 {
  for i118 = 0 to 15 {
    n81[i117, i118] := (n80[0] * w2[i117, i118]);
  }
}
for i121 = 0 to 15 {
  for i122 = 0 to 15 {
    sgd_delta_w2[i121, i122] := (w2.grad[i121, i122] + n81[i121, i122]);
  }
}

for i125 = 0 to 15 {
  for i126 = 0 to 15 {
    n79[i125, i126] := (learning_rate[0] * sgd_delta_w2[i125, i126]);
  }
}
for i129 = 0 to 15 {
  for i130 = 0 to 15 {
    w2[i129, i130] := (w2[i129, i130] - n79[i129, i130]);
  }
}
/* end */
/* w3 param sgd step */
n85[0] := 0.000200;
for i132 = 0 to 15 {
  n86[0, i132] := (n85[0] * w3[0, i132]);
}
for i134 = 0 to 15 {
  sgd_delta_w3[0, i134] := (w3.grad[0, i134] + n86[0, i134]);
}

for i136 = 0 to 15 {
  n84[0, i136] := (learning_rate[0] * sgd_delta_w3[0, i136]);
}
for i138 = 0 to 15 {
  w3[0, i138] := (w3[0, i138] - n84[0, i138]);
}
/* end */
/* end */
```

Just for scale…

SGD update step: Optimized lowered representation

```
/* scalar_loss sgd update */
/* b1 param sgd step */
learning_rate[0] := (0.000044 * (2250.000000 - i2));
for i90 = 0 to 15 {
  b1[i90] := (b1[i90] - (learning_rate[0] * (b1.grad[i90] + (0.000200 * b1[i90]))));
}
/* end */
/* b2 param sgd step */
for i98 = 0 to 15 {
  b2[i98] := (b2[i98] - (learning_rate[0] * (b2.grad[i98] + (0.000200 * b2[i98]))));
}
/* end */
/* b3 param sgd step */
b3[0] := (b3[0] - (learning_rate[0] * (b3.grad[0] + (0.000200 * b3[0]))));
/* end */
```

Much shorter.

```
/* w1 param sgd step */
for i113 = 0 to 15 {
  for i114 = 0 to 1 {
    w1[i113, i114] :=
      (w1[i113, i114] - (learning_rate[0] * (w1.grad[i113, i114] + (0.000200 * w1[i113, i114]))));
  }
}
/* end */
/* w2 param sgd step */
for i129 = 0 to 15 {
  for i130 = 0 to 15 {
    w2[i129, i130] :=
      (w2[i129, i130] - (learning_rate[0] * (w2.grad[i129, i130] + (0.000200 * w2[i129, i130]))));
  }
}
/* end */
/* w3 param sgd step */
for i138 = 0 to 15 {
  w3[0, i138] :=
    (w3[0, i138] - (learning_rate[0] * (w3.grad[0, i138] + (0.000200 * w3[0, i138]))));
}
/* end */
/* end */
```

SGD update step: compiled via C

```
/* scalar_loss sgd update */
/* b1 param sgd step */
learning_rate[0] = ((0.000044) * ((2250.000000) - (float)i2));
for (int i90 = 0; i90 <= 15; ++i90) {
  b1[i90] = (b1[i90] - (learning_rate[0] * (b1_grad[i90] + ((0.000200) * b1[i90]))));
}
/* end */
/* b2 param sgd step */
for (int i98 = 0; i98 <= 15; ++i98) {
  b2[i98] = (b2[i98] - (learning_rate[0] * (b2_grad[i98] + ((0.000200) * b2[i98]))));
}
/* end */
/* b3 param sgd step */
b3[0] = (b3[0] - (learning_rate[0] * (b3_grad[0] + ((0.000200) * b3[0]))));
/* end */
```

```
/* w1 param sgd step */
for (int i113 = 0; i113 <= 15; ++i113) {
  for (int i114 = 0; i114 <= 1; ++i114) {
    w1[i113 * 2 + i114] =
      (w1[i113 * 2 + i114] -
       (learning_rate[0] * (w1_grad[i113 * 2 + i114] + ((0.000200) * w1[i113 * 2 + i114]))));
    }
  }
/* end */
/* w2 param sgd step */
for (int i129 = 0; i129 <= 15; ++i129) {
  for (int i130 = 0; i130 <= 15; ++i130) {
    w2[i129 * 16 + i130] =
      (w2[i129 * 16 + i130] -
       (learning_rate[0] * (w2_grad[i129 * 16 + i130] + ((0.000200) * w2[i129 * 16 + i130]))));
    }
  }
/* end */
/* w3 param sgd step */
for (int i138 = 0; i138 <= 15; ++i138) {
  w3[0 * 16 + i138] =
    (w3[0 * 16 + i138] - (learning_rate[0] * (w3_grad[0 * 16 + i138] + ((0.000200) * w3[0 * 16 + i138]))));
  }
/* end */
/* end */
```

ppx_minidebug “drill-down” debugger, even w/ CUDA

- epoch = 1
- ▼<for epoch>
  - "bin/trivial_benchmark.ml":77:8
  - ▼parallel_update
    - "lib/train.ml":216:31-266:58 at time 2024-02-26 08:03:13.796623
    - ▶ grad_updates =
    - ▶ sgd_update =
    - ▶ num_devices = 8
    - ▼bindings =
      - "lib/train.ml":220:6
      - ▼<values>
        - ((static_symbol (Symbol 1)) (static_range (16)))
    - ▼all_params =
      - "lib/train.ml":225:6
      - ▶ <values>
    - ▶ merges =
    - ▶ copies = (() () () () () () ())
    - ▼round_robin
      - "lib/train.ml":133:27-155:59 at time 2024-02-26 08:03:13.8(
      - ▶ num_devices = 8
      - ▼<function -- branch 2> :: ((s, idx), :: (({static_range=None
        - "lib/train.ml":147:8-152:12 at time 2024-02-26 08:03:
        - ▼for:train:147
          - "lib/train.ml":147:8
          - i = 0

▼dev-0 run

- "arrayjit/lib/gccjit_backend.ml":506:24-533:54 at time 2024-02-26 08:03:13.955502
- global_n3[0]{=0} = 0
  - b.grad[0] := (b.grad[0] + (float *)0x55daff40add0[0]);
  - = external (float *)0x55daff40add0[0]{=0}

▶ dev-0 run
▶ dev-0 run
▶ dev-0 run
▶ dev-0 run
▶ dev-0 run
▼dev-0 run

- "arrayjit/lib/gccjit_backend.ml":506:24-533:54 at time 2024-02-26 08:03:14.765753
- scalar_loss sgd update
- b param sgd step
- ▶ global_n12[0]{=0} = 0.05625
- ▼global_n2[0]{=0.147214} -= 8.28076e-07
  - b[0] := (b[0] - (learning_rate[0] * (b.grad[0] + (0.000100 * b[0]))));
  - = (global_n12[0]{=0.05625} * (global_n3[0]{=0} + (0.0001 * global_n2[
- w param sgd step
- ▶ global_n4[0]{=0.475069} -= 2.67226e-06

▼dev-0 run

- "arrayjit/lib/gccjit_backend.ml":506:24-533:54 at time 2024-02-26 08:03:14.765956
- scalar_loss gradient update
- scalar_loss fwd
- ▶ local_n15[0]{=0} += 0.0826707
- ▶ local_n15[1]{=0} += 0.35266

Note: outdated example

`cuda-gdb` session, including CUDA source position

```
lukstafi@DESKTOP-6RRUNR4:~/ocannl$ dune build bin/moons_demo.exe
lukstafi@DESKTOP-6RRUNR4:~/ocannl$ /usr/local/cuda-12.5/bin/cuda-gdb _build/default/bin/moons_demo.exe
NVIDIA (R) cuda-gdb 12.5
Portions Copyright (C) 2007-2024 NVIDIA Corporation
Based on GNU gdb 13.2
Copyright (C) 2023 Free Software Foundation, Inc.

Reading symbols from _build/default/bin/moons_demo.exe...
(cuda-gdb) break scalar_loss_gradient_then_sgd_update.cu:252
No source file named scalar_loss_gradient_then_sgd_update.cu.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 1 (scalar_loss_gradient_then_sgd_update.cu:252) pending.
(cuda-gdb) run
Starting program: /home/lukstafi/ocannl/_build/default/bin/moons_demo.exe
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Welcome to OCANNL! Reading configuration defaults from /home/lukstafi/ocannl/ocannl_config.

CUDA thread hit Breakpoint 1, scalar_loss_gradient_then_sgd_update<<<(1,1,1),(1,1,1)>>> (
    i1=0, i2=0, scalar_loss=0x705c03000, w3=0x705c02e00, b2=0x705c02c00, w2=0x705c02800,
    moons_flat=0x705c01400, b1=0x705c01200, moons_classes=0x705c00800, b3=0x705c00600,
    w1=0x705c00400, learning_rate=0x705c00200) at scalar_loss_gradient_then_sgd_update.cu:252
252       b3[0] = (b3[0] - (learning_rate[0] * (b3_grad[0] + ((0.000200) * b3[0]))));
(cuda-gdb) print b3_grad[0]
$1 = 0.50000006
(cuda-gdb) print learning_rate[0]
$2 = 0.0989999995
```

Deep dive: gradient update

Defining operations

TODO

Gradient update step

Gradient updated step: Optimized lowered representation

Example: GPT

This will remain a TODO for a long time :-(

Soft deadline by September but not guaranteed.

Deep dive: multi-device data-parallelism

SGD update step: compiled via CUDA

TODO

Multi-device data-parallel training

TODO

Benchmarking parallelization options

TODO

ppx_minidebug traces from devices in separate files

TODO

Deep dive: CUDA backend

TODO