Package ‘tfautograph’

October 14, 2022

Title Autograph R for 'Tensorflow'

Version 0.3.2

Description Translate R control flow expressions into 'Tensorflow' graphs.

SystemRequirements TensorFlow (https://www.tensorflow.org/)

URL `https://t-kalinowski.github.io/tfautograph/`

BugReports `https://github.com/t-kalinowski/tfautograph/issues`

Depends R (>= 3.1)

Imports reticulate, backports

License GPL-3

Encoding UTF-8

RoxygenNote 7.1.2

Suggests rlang, tensorflow, testthat (>= 2.1.0)

Language en-US

NeedsCompilation no

Author Tomasz Kalinowski [aut, cre]

Maintainer Tomasz Kalinowski <`kalinowskit@gmail.com`>

Repository CRAN

Date/Publication 2021-09-17 20:30:02 UTC

R topics documented:

ag_if_vars — *Specify* `tf.cond()` *output structure when autographing* `if`

Description

This function can be used to specify the output structure from `tf.cond()` when autographing an `if` statement. In most use cases, use of this function is purely optional. If not supplied, the `if` output structure is automatically built.

Usage

```
ag_if_vars(
  ...,
  modified = list(),
  return = FALSE,
  undefs = NULL,
  control_flow = 0
)
```

Arguments

`...`	Variables modified by the `tf.cond()` node supplied as bare symbols like `foo` or expressions using `$` e.g, `foo$bar`. Symbols do not have to exist before the autographed `if` so long as they are created in both branches.
`modified`	Variables names supplied as a character vector, or a list of character vectors if specifying nested complex structures. This is an escape hatch for the lazy evaluation semantics of `...`
`return`	logical, whether to include the return value the evaluated R expression in the `tf.cond()`. if `FALSE` (the default), only the objects assigned in scope are captured.
`undefs`	A bare character vector or a list of character vectors. Supplied names are exported as undefs in the parent frame. This is used to give a more informative error message when attempting to access a variable that can't be balanced between branches.
`control_flow`	An integer, the maximum number of control-flow statements (`break` and/or `next`) that will be captured in a single branch as part of the `tf.cond()`. Do not count statements in loops that are dispatching to standard R control flow (e.g., don't count `break` statements in a `for` loop that is iterating over an R vector)

Details

If the output structure is not explicitly supplied via `ag_if_vars()`, then the output structure is automatically composed: The true and false branches of the expression are traced into concrete functions, then the output signature from the two branch functions are balanced. Balancing is performed by either fetching a variable from an outer scope or by reclassifying a symbol as an undef.

When dealing with complex composites (that is, nested structures where a modified tensor is part of a named list or dictionary), care is taken to prevent unnecessarily capturing other unmodified tensors in the structure. This is done by pruning unmodified tensors from the returned output structure, and then merging them back with the original object recursively. One limitation of the implementation is that lists must either be fully named with unique names, or not named at all, partially named lists or duplicated names in a list throw an error. This is due to the conversion that happens when going between python and R: named lists get converted to python dictionaries, which require that all keys are unique. Additionally, pruning of unmodified objects from an autographed `if` is currently only supported for named lists (python dictionaries). Unnamed lists or tuples are passed as is (e.g, no pruning and merging done), which may lead to unnecessarily bloat in the constructed graphs.

Value

`NULL`, invisibly

Examples

```
## Not run:
# these examples only have an effect in graph mode
# to enter graph mode easily we'll create a few helpers
ag <- autograph

# pass which symbols you expect to be modifed or created liks this:
ag_if_vars(x)
ag(if (y > 0) {
  x <- y * y
} else {
  x <- y
})

# if the return value from the if expression is important, pass `return = TRUE`
ag_if_vars(return = TRUE)
x <- ag(if(y > 0) y * y else y)

# pass complex nested structures like this
x <- list(a = 1, b = 2)

ag_if_vars(x$a)
ag(if(y > 0) {
  x$a <- y
})

# undefs are for mark branch-local variables
ag_if_vars(y, x$a, undef = "tmp_local_var")
```

```
ag(if(y > 0) {
  y <- y * 100
  tmp_local_var <- y + 1
  x$a <- tmp_local_var
})

# supplying `undef` is not necessary, it exists purely as a way to supply a
# guardrail for defensive programming and/or to improve code readability

## modified vars can be supplied in `...` or as a named arg.
## these paires of ag_if_vars() calls are equivalent
ag_if_vars(y, x$a)
ag_if_vars(modified = list("y", c("x", "a")))

ag_if_vars(x, y, z)
ag_if_vars(modified = c("x", "y", "z"))


## control flow
# count number of odds between 0:10
ag({
  x <- 10
  count <- 0
  while(x > 0) {
    ag_if_vars(control_flow = 1)
    if(x %% 2 == 0)
      next
    count <- count + 1
  }
})

## End(Not run)
```

ag_loop_vars *Specify loop variables*

Description

This can be used to manually specify which variables are to be included explicitly as `loop_vars` when autographing an expression into a `tf.while_loop()` call, or the `loop_vars` equivalent when building a `dataset.reduce()`.

Usage

```
ag_loop_vars(
  ...,
  list = character(),
  include = character(),
  exclude = character(),
  undef = character()
)
```

Arguments

`...`	Variables as bare symbol names
`list, include, exclude`	optionally, the variable names as a character vector (use this as an escape hatch from the `...` lazy evaluation semantics).
`undef`	character vector of symbols

Details

Use of this is usually not required as the loop variables are automatically inferred. Inference is done by statically looking through the loop body and finding the symbols that are the targets of the common assignment operators from base R (`<-`, `->`, `=`), from package:zeallot (`%<-%` and `%->%`) and package:magrittr (`%<>%`).

In certain circumstances, this approach may capture variables that are intended to be local variables only. In those circumstances it is also possible to specify them preceded with a `-`.

Note, the specified loop vars are expected to exist before the autographed expression, and a warning is issued otherwise (usually immediately preceding an error thrown when attempting to actually autograph the expression)

Only bare symbol names can be supplied as loop vars. In the future, support may be expanded to allow for nested complex composites (e.g., specifying variables that are nested within a more complex structure–passing `ag_loop_vars(foo$bar$baz)` is currently not supported.)

Value

the specified hint invisibly.

Note

The semantics of this function are inspired by base::rm()

Examples

```
## Not run:
i <- tf$constant(0L)

autograph({
  ag_loop_vars(x, i)
  while(x > 0) {
    if(x %%2 == 0)
      i <- i + 1L
    x <- x - 1
  }
})

## sometimes, a variable is infered to be a loop_var unnecessarily. For example
x <- tf$constant(1:10)

# imagine x is left over in the current scope from some previous calculations
# It's value is not important, but it exists
```

```
autograph({
  for(i in tf$constant(1:6)) {
    x <- i * i
    tf$print(x)
  }
})

# this will throw an error because `x` was infered to be a `loop_var`,
# but it's shape witin the loop body is different from what it was before.
# there are two solutions to prevent `x` from being captured as a loop_var
## 1) remove `x` from the current scope like so:
rm(x)

## 2) provide a hint like so:
ag_loop_vars(-x)

## if your variable names are being dynamically generated, there is an
## escape hatch for the lazy evaluation semantics of ...
ag_loop_vars(exclude = "x")

## End(Not run)
```

ag_name *Specify a tensor name*

Description

This can be used before any autographed expression that results in the creation of a tensor or op graph node. This can be used before `for` (both with tensors and datasets), `while`, and `if` statements.

Usage

```
ag_name(x)
```

Arguments

x	A string

Value

x, invisibly

Examples

```
## Not run:
## when you're in graph mode. (e.g, tf$executing_eagerly == FALSE)

ag_name("main-training-loop")
for(elem in dataset) ...

## End(Not run)
```

`ag_while_opts` *specify* `tf.while_loop` *options*

Description

See https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/while_loop for additional details.

Usage

```
ag_while_opts(
  ...,
  shape_invariants = NULL,
  parallel_iterations = 10L,
  back_prop = TRUE,
  swap_memory = FALSE,
  maximum_iterations = NULL
)
```

Arguments

`...`	Ignored, used to ensure all arguments supplied are named.
`shape_invariants`	The shape invariants for the loop variables.
`parallel_iterations`	The number of iterations allowed to run in parallel. It must be a positive integer.
`back_prop`	Deprecated (optional). `FALSE` disables support for back propagation. Prefer using `tf$stop_gradient` instead.
`swap_memory`	Whether GPU-CPU memory swap is enabled for this loop.
`maximum_iterations`	Optional maximum number of iterations of the while loop to run. If provided, the cond output is AND-ed with an additional condition ensuring the number of iterations executed is no greater than `maximum_iterations`.

Value

‘NULL“ invisibly, called for it’s side effect.

Note

Use `ag_name()` to supply `name` and `ag_loop_vars()` to supply `loop_vars` directly.

This is only applicable when autograph in graph mode, otherwise this has no effect.

Examples

```
## Not run:
## use tf_function() to enter graph mode:
tf_function(autograph(function(n) {
  ag_name("silly-example")
  ag_while_opts(back_prop = FALSE)
  while(n > 0)
    n <- n - 1
}))

## End(Not run)
```

---

autograph *Autograph R code*

Description

Note, this documentation page is meant to serve as a technical reference, not an introduction to `autograph`. For the latter, please visit the documentation website: (https://t-kalinowski.github.io/tfautograph/) or see the package vignettes.

Usage

```
autograph(x)
```

Arguments

`x`	a function supplied as a bare symbol, or an expression

Value

if `x` is a function, then the the same function with a new parent environment, `package:tfautograph:ag_mask`, which is the autograph mask that contains implementations of R control flow primitives that are capable of handling tensorflow tensors. The parent of the `package:tfautograph:ag_mask` in turn is the original environment of `x`.

if `x` is an expression, then that expression is evaluated in a special environment with the autograph mask `ag_mask` active. If the result of that expression included local assignment or modifications of variables, (for example, via `<-`), those modified variables are then exported into the current frame. The return value of the expression is then returned.

tf_assert *tf_assert*

Description

A thin wrapper around `tf$Assert()` that automatically constructs an informative error message (passed on to `data` argument), which includes the expression passed to `condition`, the values of the symbols found in the expression, as well as the full R call stack at the time the `tf$Assert()` node is created.

Usage

```
tf_assert(
  condition,
  ...,
  expr = substitute(condition),
  summarize = NULL,
  name = NULL
)
```

Arguments

`condition`	A boolean tensor
`...`	Additional elements passed on to `data`. (e.g, an informative error message as a string, additional tensor values that might be useful to have in the error message, etc.)
`expr`	A language object, provided in case `condition` is already computed prior to the call
`summarize`	Print this many entries of each tensor.
`name`	A name for this operation (optional).

Examples

```
## Not run:
x <- tf$constant(-1)
try(tf_assert(x > 0, "oopsies! x must be greater than 0"))

## End(Not run)
```

`tf_case` *tf.case*

Description

This is a minimal wrapper around `tf.case()` that allows you to supply the `pred_fn_pairs` using the ~.

Usage

```
tf_case(
  ...,
  pred_fn_pairs = list(...),
  default = NULL,
  exclusive = FALSE,
  name = "case"
)
```

Arguments

`..., pred_fn_pairs`	a list `pred_fn_pairs` supplied with the ~ like so: `pred` ~ `fn_body`
`default`	a function, optionally specified with the ~, (or something coercible to a function via `as.function()`)
`exclusive`	bool, whether to evaluate all `preds` and ensure only one is true. If `FALSE` (the default), then the `preds` are evaluated in the order supplied until the first `TRUE` value is encountered (effectively, acting as an `if()... else if() ... else if() ...` chain)
`name`	a string, passed on to `tf.case()`

Value

The result from `tf$case()`

Examples

```
## Not run:
fizz_buzz_one <- function(x) {
  tf_case(
    x %% 15 == 0 ~ "FizzBuzz",
    x %%  5 == 0 ~ "Buzz",
    x %%  3 == 0 ~ "Fizz",
    default = ~ tf$as_string(x, precision = 0L)
  )
}

fn <- tf_function(autograph(function(n) {
  for(e in tf$range(n))
    tf$print(fizz_buzz_one(e))
}))

x <- tf$constant(16)
fn(x)

## End(Not run)
```

`tf_cond` *tf.cond*

Description

This is a minimal wrapper around `tf$cond()` that allows you to supply `true_fn` and `false_fn` as lambda functions defined using the tilde ~.

Usage

```
tf_cond(pred, true_fn, false_fn, name = NULL)
```

Arguments

`pred`	R logical or a tensor.
`true_fn, false_fn`	a ~ function, a function, or something coercible to a function via `as.function`
`name`	a string, passed on to `tf.cond()`

Value

if cond is a tensor, then the result of `tf.cond()`. Otherwise, if `pred` is an `EagerTensor` or an R logical, then the result of either `true_fn()` or `false_fn()`

Note

in Tensorflow version 1, the `strict` keyword argument is supplied with a value of `TRUE` (different from the default)

Examples

```
## Not run:
## square if positive
# using tf$cond directly:
raw <- function(x) tf$cond(x > 0, function() x * x, function() x)

# using tf_cond() wrapper
tilde <- function(x) tf_cond(x > 0, ~ x * x, ~ x)

## End(Not run)
```

tf_map tf.map_fn()

Description

Thin wrapper around `tf.map_fn()` with the following differences:

- accepts `purrr` style ~ lambda syntax to define function `fn`.
- The order of `elems` and `fn` is switched to make it more pipe `%>%` friendly and consistent with R mappers `lapply()` and `purrr::map()`.

Usage

```
tf_map(
  elems,
  fn,
  dtype = NULL,
  parallel_iterations = NULL,
  back_prop = TRUE,
  swap_memory = FALSE,
  infer_shape = TRUE,
  name = NULL
)
```

Arguments

`elems`	A tensor or (possibly nested) sequence of tensors, each of which will be unpacked along their first dimension. The nested sequence of the resulting slices will be applied to `fn`.
`fn`	An R function, specified using `purrr` style ~ syntax, a character string, a python function (or more generally, any python object with a `__call__` method) or anything coercible via `as.function()`. The function will be be called with one argument, which will have the same (possibly nested) structure as `elems`. Its output must return the same structure as `dtype` if one is provided, otherwise it must return the same structure as `elems`.
`dtype`	(optional) The output type(s) of fn. If fn returns a structure of Tensors differing from the structure of elems, then dtype is not optional and must have the same structure as the output of fn.
`parallel_iterations`	(optional) The number of iterations allowed to run in parallel. When graph building, the default value is 10. While executing eagerly, the default value is set to 1.
`back_prop`	(optional) True enables support for back propagation.
`swap_memory`	(optional) True enables GPU-CPU memory swapping.
`infer_shape`	(optional) False disables tests for consistent output shapes.
`name`	(optional) Name prefix for the returned tensors.

Value

A tensor or (possibly nested) sequence of tensors. Each tensor packs the results of applying fn to tensors unpacked from elems along the first dimension, from first to last.

---

tf_switch *tf.switch_case*

Description

tf.switch_case

Usage

```
tf_switch(
  branch_index,
  ...,
  branch_fns = list(...),
  default = NULL,
  name = "switch_case"
)
```

Arguments

`branch_index`	an integer tensor
`..., branch_fns`	a list of function bodies specified with a ~, optionally supplied with a branch index on the left hand side. See examples
`default`	A function defined with a ~, or something coercible via ‘as.function()“
`name`	a string, passed on to `tf.switch_case()`

Value

The result from `tf.switch_case()`

Examples

```
## Not run:
tf_pow <- tf_function(function(x, pow) {
   tf_switch(pow,
   0 ~ 1,
   1 ~ x,
   2 ~ x * x,
   3 ~ x * x * x,
   default = ~ -1)
})

# can optionally also omit the left hand side int, in which case the order of
# the functions is used.
tf_pow <- function(x, pow) {
  tf_switch(pow,
            ~ 1,
            ~ x,
            ~ x * x,
            ~ x * x * x,
            default = ~ -1)
}

# supply just some of the ints to override the default order
tf_pow <- function(x, pow) {
  tf_switch(pow,
            3 ~ x * x * x,
            2 ~ x * x,
            ~ 1,
            ~ x,
            default = ~ -1)
}

# A slightly less contrived example:
tf_norm <- tf_function(function(x, l) {
  tf_switch(l,
            0 ~ tf$reduce_sum(tf$cast(x != 0, tf$float32)), # L0 norm
            1 ~ tf$reduce_sum(tf$abs(x)),                   # L1 norm
            2 ~ tf$sqrt(tf$reduce_sum(tf$square(x))),       # L2 norm
            default = ~ tf$reduce_max(tf$abs(x)))         # L-infinity norm
})

## End(Not run)
```

view_function_graph *Visualizes the generated graph*

Description

Visualizes the generated graph

Usage

```
view_function_graph(
  fn,
  args,
  ...,
  name = deparse(substitute(fn)),
  profiler = FALSE,
  concrete_fn = do.call(fn$get_concrete_fn, args),
  graph = concrete_fn$graph
)
```

Arguments

`fn`	TensorFlow function (returned from `tf.function()`)
`args`	arguments passed to `fun`
`...`	other arguments passed to `tensorflow::tensorboard()`
`name`	string, provided to tensorboard
`profiler`	logical, passed on to `tf.compat.v2.summary.trace_on()` (only used in eager mode)
`concrete_fn`	a `ConcreteFunction` (only used in graph mode, ignored with a warning if executing eagerly)
`graph`	a tensorflow graph (only used in graph mode, ignored with a warning if executing eagerly)

Examples

```
## Not run:
fn <- tf_function(function(x) autograph(if(x > 0) x * x else x))
view_function_graph(fn, list(tf$constant(5)))

## End(Not run)
```

`[[<-.tensorflow.python.ops.tensor_array_ops.TensorArray` `TensorArray.write()`

Description

`TensorArray.write()`

Usage

```
## S3 replacement method for class 'tensorflow.python.ops.tensor_array_ops.TensorArray'
ta[[i, ..., name = NULL]] <- value
```

Arguments

`ta`	a tensorflow `TensorArray`
`i`	something castable to an int32 scalar tensor. 0-based.
`...`	Error if anything is passed to `...`
`name`	A scalar string, name of the op
`value`	The value to write.

Examples

```
## Not run:
ta <- tf$TensorArray(tf$float32, size = 5L)
for(i in 0:4)
  ta[[i]] <- i
ta$stack()

# You can use this to grow objects in graph mode
accuracies_log <- tf$TensorArray(tf$float32, size = 0L, dynamic_size=TRUE)
for(epoch in 0:4)
  accuracies_log[[epoch]] <- runif(1)
acc <- accuracies_log$stack()
acc

## End(Not run)
```

Index