PlaidML Documentation

Vertex.AI

Apr 11, 2018

Contents

A framework for making deep learning work everywhere.

PlaidML is a multi-language acceleration framework that:

- Enables practitioners to deploy high-performance neural nets on any device
- Allows hardware developers to quickly integrate with high-level frameworks
- Allows framework developers to easily add support for many kinds of hardware
- Works on all major platforms - Linux, macOS, Windows

For background and early benchmarks see our blog post announcing the release. PlaidML is under active development and should be thought of as alpha quality.

CHAPTER 1

Ubuntu Linux

If necessary, install Python's 'pip' tool.

```
sudo add-apt-repository universe && sudo apt update
sudo apt install python-pip
```

Make sure your system has OpenCL.

```
sudo apt install clinfo
clinfo
```

If clinfo reports "Number of platforms" == 0, you must install a driver.

If you have an NVIDIA graphics card:

```
sudo add-apt-repository ppa:graphics-drivers/ppa && sudo apt update
sudo apt install nvidia-modprobe nvidia-384 nvidia-opencl-icd-384 libcuda1-384
```

If you have an AMD card, download the AMDGPU PRO driver and install according to AMD's instructions.

Best practices for python include judicious usage of Virtualenv, and we certainly recommend creating one just for plaidml:

```
virtualenv plaidml-venv
source ./plaidml-venv/bin/activate
pip install -U plaidml-keras
```

Alternatively, install the PlaidML wheels system-wide:

```
sudo -H pip install -U plaidml-keras
```

Next, setup PlaidML to use your preferred computing device:

```
plaidml-setup
```

You can test your installation by running MobileNet in plaidbench: (Remember to use sudo -H if you're not using a Virtualenv)

```
git clone https://github.com/plaidml/plaidbench.git
cd plaidbench
pip install -r requirements.txt
python plaidbench.py mobilenet
```

You can adapt any Keras code by using the PlaidML backend instead of the TensorFlow, CNTK, or Theano backend that you normally use.

Simply insert this code **BEFORE you `import keras`**:

```
# Install the plaidml backend
import plaidml.keras
plaidml.keras.install_backend()
```

CHAPTER 2

Building and Testing PlaidML

PlaidML depends on Bazel v0.11.0 or higher.

2.1 Building PlaidML

2.1.1 Install Requirements

pip install -r requirements.txt

2.1.2 Linux

```
bazel build --config linux_x86_64 plaidml:wheel plaidml/keras:wheel
sudo pip install -U bazel-bin/plaidml/*whl bazel-bin/plaidml/keras/*whl
```

2.1.3 macOS

The versions of bison and flex that are provided with xcode are too old to build PlaidML. It's easiest to use homebrew to install all the prerequisites:

```
brew install bazel bison flex
```

Then, use bazel to build, sepecifying the correct config from tools/bazel.rc:

```
bazel build --config macos_x86_64 plaidml:wheel plaidml/keras:wheel
sudo pip install -U bazel-bin/plaidml/*whl bazel-bin/plaidml/keras/*whl
```

2.2 Testing PlaidML

Unit tests are executed through bazel:

```
bazel test ...
```

Unit tests for frontends are marked manual and must be executed individually (requires running `plaidml-setup` prior to execution)

```
bazel test plaidml/keras:backend_test
```

CHAPTER 3

PlaidML Architecture Overview

At a High Level PlaidML Consists of:

- A core that exposes a C and C++ API:
  - A HAL API and a library of backends that implement it (OpenCL/LLVM/etc)
  - A runtime which takes tile code, optimizes it based on parameters from the HAL, and a Platform that schedules operations and memory layout based on the type of Platform (Local / Remote)
- Python bindings built on top of the C API
  - An operations library which is a generic library of tile code
  - An API that can be called directly or used to develop other frontends
- Frontend adapters that utilize the op library and the API to implement support for that frontend
  - ONNX
  - Keras

CHAPTER 4

Tile Op Tutorial

Sometimes, when implementing a cutting-edge or unusual network, the functions you need are not available in Keras. This tutorial will show you how to write a new operation in the PlaidML backend to Keras.

Accompanying the instructions will be a demonstration of how these techniques can be used to write the Keras backend function `categorical_crossentropy`.

4.1 Write a Simple Case in Tile

Plaid performs machine learning operations by executing code written in the Tile language. Tile is useful for being relatively easy to translate into both optimized GPU code and also mathematical notation, but it is not well-suited to generalizing similar operations. For instance, the same operation applied to a tensor with a different number of dimensions must be written with different Tile code.

It is often easiest to first write a simple case in Tile, for which PlaidML and Keras serve as a thin wrapper. Tile code is covered in [[another tutorial|Tile-Tutorial]], so we'll keep the explanation of the Tile code for categorical crossentropy to a minimum.

As our simple case for `categorical_crossentropy`, we'll assume that the `target` and `output` tensors are 2 dimensional. In this case, the mathematical expression of categorical crossentropy is

A key feature of this formula is the summation over the index `y`. In Tile, such operations cannot typically be performed on the same line as elementwise operations, so we will split this formula into three parts:

These formulas correspond quite directly with the Tile code for this operation.

```
function (T[X, Y], O[X, Y]) -> (R) {
    LO = log(O);
    Temp[x: X] = +(LO[x, y] * T[x, y])
    R = -Temp;
}
```

4.2 Wrap the Tile Code

Keras can't interpret Tile code, but the PlaidML backend can. The class `_Op` (in our backend code `plaidml/keras/backend.py`) can be initialized with Tile code and will produce an object Keras can interact with as a Keras tensor. It requires several additional parameters as well:

- **ident**: A string giving the name of the operation. Appears in `__repr__` and `__str__`.
- **dtype**: The data type of the result tensor, e.g. `'float32'`.
- **shape**: The Keras shape of the result tensor. A tuple of positive integers and/or Nones.
- **code**: Tile code for the operation, given as a string.
- **inputs**: An ordered dictionary. The key is the name of the input as a string; the value is the tensor to be input. Be sure the name of the first (and second, etc) input here is the same as the name of the first input in the Tile code.
- **outputs**: A list of strings giving the names of the output(s). Please ensure the name(s) (and, if applicable, order) of the output(s) match(es) the Tile code.

We can now write the Python code for this case of categorical crossentropy! It won't work if we get tensors of the wrong dimension, but in the 2D `from_logits=False` case, this is sufficient:

```
def categorical_crossentropy(target, output):
    f = """function (T[X, Y], O[X, Y]) -> (R) {
                LO = Log(O);
                Temp[x: X] = +(LO[x, y] * T[x, y]);
                R = -Temp;
            }"""
    return _Op('cat_xentropy', output.dtype, output.shape[:-1], f,
                OrderedDict([('T', target), ('O', output)]), ['R'])
```

Note one parameter that isn't needed: Tile code for the gradient. PlaidML includes an autodiff utility for Tile that Keras can invoke to produce and evaluate Tile code for the gradient of any Tile function. You only need to write forward-pass operations; Keras and PlaidML will work out their gradients automatically.

4.3 Generalize

Having written Tile code for one case (crossentropy in 2D tensors) we generalize to all the cases we want the function to handle (crossentropy in arbitrary-dimensional tensors; also accept logits). Tile is not designed for this sort of generalization, so we change what Tile code we write using substitution and string manipulation.

For categorical crossentropy, we change the number of dimensions by adding (or removing) dimension sizes `X` and corresponding indices `x`. We want a number of each equal to `target.ndim - 1`, so we write the following:

```
fixed_dims == ", ".join(["X{}.format(i) for i in range(target.ndim - 1)])
fixed_idxs == ", ".join(["x{}.format(i) for i in range(target.ndim - 1)])
```

We substitute these into the Tile code using the Python string `format` function:

```
f = """function (T[{fixed_dims}, Y], O[{fixed_dims}, Y]) -> (R) {
            LO = Log(O);
            Temp[{fixed_idxs}: {fixed_dims}] = +(LO[{fixed_idxs}, y] * T[{fixed_idxs},␣
↪y]);
            R = -Temp;
        }""".format(fixed_dims=fixed_dims, fixed_idxs=fixed_idxs)
```

We could handle `from_logits` by manipulating the Tile code in a similar way. However, that case merely requires performing a softmax first, and softmax is already defined in the backend! So we instead add python code

```
if from_logits:
    output = softmax(output)
```

Putting it all together, we have

```
def categorical_crossentropy(target, output):
    if from_logits:
        output = softmax(output)
    fixed_dims == ", ".join(["X{}.format(i) for i in range(target.ndim - 1)])
    fixed_idxs == ", ".join(["x{}.format(i) for i in range(target.ndim - 1)])
    f = """function (T[{fixed_dims}, Y], O[{fixed_dims}, Y]) -> (R) {
                LO = Log(O);
                Temp[{fixed_idxs}: {fixed_dims}] = +(LO[{fixed_idxs}, y] * T[{fixed_
↪idxs}, y]);
                R = -Temp;
            }""".format(fixed_dims=fixed_dims, fixed_idxs=fixed_idxs)
    return _Op('cat_xentropy', output.dtype, output.shape[:-1], f,
                OrderedDict([('T', target), ('O', output)]), ['R'])
```

4.4 Add Tests, Handle Edge Cases

If you were to test the above code, you would find that it worked great . . . except if you passed it 1D tensors. That's mostly fine (especially in Keras where nearly everything is batched), but "mostly fine" will come back to haunt you, so you should handle that edge case (this is left as an exercise for the reader). If you compare to the backend code we actually use for this, you'll see that we also preprocess `output` if `from_logits` is False but the input is not directly from softmax. This won't change `output` if it is formatted like the result of a softmax, but it will prevent domain errors from log that can occur if someone (improperly) passes the function a non-softmaxed tensor.

4.5 Wrap with Keras Code

For `categorical_crossentropy`, we're done: this is a standard Keras backend function and Keras will use it where it needs it. If you are creating a novel operation, however, you may want to wrap this backend function in a higher-level Keras object. For details on how to do this, see the Keras documentation.

For some operations this is unnecessary. You can always call your custom backend function directly among your other Keras code, just like you can call a TensorFlow function directly if you're using the TensorFlow backend.

CHAPTER 5

PlaidML

A framework for making deep learning work everywhere.

PlaidML is a multi-language acceleration framework that:

- Enables practitioners to deploy high-performance neural nets on any device
- Allows hardware developers to quickly integrate with high-level frameworks
- Allows framework developers to easily add support for many kinds of hardware

For more information, see the PlaidML Announcement, and the PlaidML GitHub Repository.

5.1 About this module

This module provides the low-level PlaidML Python API.

Using this API directly requires either knowledge of the Tile language (used to describe the computations that make up a neural network), or a pre-built serialized network (which encapsulates the Tile operations that define the shape of the network, and the intra-network connection weights found by training the network).

5.2 Higher-level APIs

plaidml.keras - Integration with the Keras machine learning framework. This is useful for easily describing and training neural networks.

plaidml.tile - Utilities for building up composite TILE functions from high-level operation semantics.

5.3 Modules

keras	Patches in a PlaidML backend for Keras.
op	The TILE standard operation library.
tile	TILE program construction utilities.
exceptions	

5.3.1 plaidml.keras

Description

Patches in a PlaidML backend for Keras.

This module hooks the system meta module path to add a backend for Keras that uses PlaidML for computation. The actual backend is implemented in backend.py.

To use this module to install the PlaidML backend:

```
import plaidml.keras
plaidml.keras.install_backend()
```

This should be done in the main program module, after `__future__` imports (if any) and before importing any Keras modules. Calling `install()` replaces the standard keras.backend module with plaidml.keras.backend, causing subsequently loaded Keras modules to use PlaidML.

You can explicitly set the installed backend via the environment:

PLAIDML_KERAS_BACKEND: Selects the backend to use. If this is not set, the standard PlaidML backend is used. Possible values are "plaidml" and "theano".

You can also explicitly pass the backend in the call to `install_backend()`.

(As an aside: we don't use the standard Keras approach of having you edit `~/.keras/keras.json` to set the backend, because we want code that doesn't patch in the PlaidML backend loader to continue to work. If Keras ever does support dynamic loading of backends that aren't hard-coded into Keras, we will switch to that mechanism.)

Functions

install_backend([import_path, backend, ...])	Installs the PlaidML backend loader, overriding the default keras.

install_backend

plaidml.keras.**install_backend**(*import_path='keras.backend'*, *backend='plaidml'*, *trace_file=None*)

Installs the PlaidML backend loader, overriding the default keras.backend.

Parameters

- **import_path** – The name of the module to patch.
- **backend** – The name of the backend to patch in.
- **trace_file** – A file object to write trace data to. This may also be the name of a file, which will be opened with mode 'w' (clobbering the existing file, if any).

5.3.2 plaidml.op

Description

The TILE standard operation library.

These operations have been shown to be useful across a variety of frameworks. (Frameworks are of course free to define their own operations in addition to these, although it'll be easier to use them with these if a framework's own operations are defined using the standard *plaidml.tile* base classes.)

Each operation is defined as a `tile.Operation` subclass, allowing it to be used in pattern matching. Additionally, each operation is provided via a top-level function that wraps the class, allowing composite operations to be built up using a functional programming style.

See the PlaidML Op Tutorial for information about writing your own custom operations.

Classes

ArgMax(value[, axis])	Maximum of elements along an axis.
AutoPadding	
AveragePool(data, kernel_shape, pads, strides)	A standard ML average pooling operator.
BinaryCrossentropy(target, output, epsilon)	Computes the binary crossentropy of a value relative to a target.
Cast(x, dtype)	
ClipMax(value, max_val)	Clips a Value to a maximum bound.
ClipMin(value, min_val)	Clips a Value to a minimum bound.
Concatenate(tensors[, axis])	Concatenates tensors to make a single larger tensor.
Convolution(data, kernel[, strides, ...])	A standard ML convolution operator.
ConvolutionDataFormat	
ConvolutionTranspose(x, kernel, ...)	A transposed convolution operator.
CumulativeSum(x[, axis])	Cumulative sum of a tensor
Dot(x, y)	Dot-product of two tensors.
Elu(x[, alpha])	Exponential linear unit.
Equal(lhs, rhs)	Elementwise tensor equality.
Equal_ArgMax(lhs, rhs)	
Flatten(data)	Flattens a tensor to a one-dimensional value.
Gather(value, indicies)	Gathers elements of a tensor.
Gemm(a, b, c[, alpha, beta, broadcast, ...])	Implements a general matrix multiplication.
Gradients(loss, variables)	Compute the gradients of a loss with respect to a set of values
Hardmax(data)	Implements a standard ML hardmax.
Identity(x)	A simple identity operation.
IsMax(value, axes)	True iff an input's value is the maximum along some set of axes.
LogSoftmax(data)	Implements the log() of a standard ML softmax.
MatMul(a, b)	A matrix multiplication, using numpy semantics.
MaxPool(data, padding, kernel_shape, pads, ...)	A standard ML max pooling operator.
MaxReduce(x[, axes, keepdims])	Computes the maximum value along some set of axes.
Mean(x[, axes, keepdims, floatx])	Computes the mean value along some set of axes.
MinReduce(x[, axes, keepdims])	Computes the minimum value along some set of axes.
NotEqual(lhs, rhs)	Elementwise tensor inequality.

Continued on next page

Table 3 – continued from previous page

Pow(x, p)	An elementwise pow() function.
Prod(value[, axes, keepdims, floatx])	
Relu(x[, alpha, max_value])	A Rectified Linear Unit.
Reshape(x, dims)	Reshapes a tensor, without changing the type or number of elements.
SliceTensor(data[, axes, ends, starts])	Implements tensor slicing.
Softmax(data)	Implements a standard ML softmax.
Sqrt(x)	Computes the elementwise square root of a value.
Summation(value[, axes, keepdims, floatx])	Sums an input value along some set of axes.
Variance(x[, axes, keepdims, floatx])	

ArgMax

class plaidml.op.**ArgMax**(*value*, *axis=-1*)

Maximum of elements along an axis.

Builds a tensor whose elements are the maximum value on some axis of an input tensor.

Methods

__init__(value[, axis])	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

AutoPadding

class plaidml.op.**AutoPadding**

Attributes

EXPLICIT
SAME_LOWER
SAME_UPPER
VALID

AveragePool

class plaidml.op.**AveragePool**(*data*, *kernel_shape*, *pads*, *strides*, *padding=<AutoPadding.EXPLICIT: 1>*)

A standard ML average pooling operator.

Methods

__init__(data, kernel_shape, pads, strides)	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.

Continued on next page

Table 6 – continued from previous page

function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

BinaryCrossentropy

class plaidml.op.**BinaryCrossentropy**(*target*, *output*, *epsilon*, *from_logits=False*)

Computes the binary crossentropy of a value relative to a target.

Methods

__init__(target, output, epsilon[, from_logits])	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

Cast

class plaidml.op.**Cast**(*x*, *dtype*)

Methods

__init__(x, dtype)	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

ClipMax

class plaidml.op.**ClipMax**(*value*, *max_val*)

Clips a Value to a maximum bound.

Methods

__init__(value, max_val)	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

ClipMin

class plaidml.op.**ClipMin**(*value*, *min_val*)

Clips a Value to a minimum bound.

Methods

__init__(value, min_val)	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

Concatenate

class plaidml.op.**Concatenate**(*tensors*, *axis=-1*)

Concatenates tensors to make a single larger tensor.

Methods

__init__(tensors[, axis])	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

Convolution

class plaidml.op.**Convolution**(*data*, *kernel*, *strides=None*, *padding=<AutoPadding.EXPLICIT: 1>*, *pads=None*, *group=1*, *kernel_shape=None*, *data_format=None*, *dilation_rate=None*, *channelwise=False*)

A standard ML convolution operator.

Methods

__init__(data, kernel[, strides, padding, ...])	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

ConvolutionDataFormat

class plaidml.op.**ConvolutionDataFormat**

Attributes

CHANNELS_FIRST
CHANNELS_LAST

ConvolutionTranspose

class plaidml.op.**ConvolutionTranspose**(*x*, *kernel*, *output_shape*, *strides*, *padding*, *data_format*)

A transposed convolution operator.

Methods

__init__(x, kernel, output_shape, strides, ...)	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

CumulativeSum

class plaidml.op.**CumulativeSum**(*x*, *axis=0*)

Cumulative sum of a tensor

Methods

__init__(x[, axis])	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

Dot

class plaidml.op.**Dot**(*x*, *y*)

Dot-product of two tensors.

Methods

__init__(x, y)	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

Elu

class plaidml.op.**Elu**(*x*, *alpha=1.0*)

Exponential linear unit.

Methods

__init__(x[, alpha])	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

Equal

class plaidml.op.**Equal**(*lhs*, *rhs*)

Elementwise tensor equality.

Builds a boolean tensor whose values are true where the corresponding elements of the inputs are equal.

Methods

__init__(lhs, rhs)	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

Equal_ArgMax

class plaidml.op.**Equal_ArgMax**(*lhs*, *rhs*)

Methods

__init__(lhs, rhs)	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

Flatten

class plaidml.op.**Flatten**(*data*)

Flattens a tensor to a one-dimensional value.

Methods

__init__(data)	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

Gather

class plaidml.op.**Gather**(*value*, *indicies*)

Gathers elements of a tensor.

Methods

`__init__`(value, indicies)	*Operation* constructor.
`bind`(bindings)	Builds an output variable dictionary for the operation.
`function`(*args, **kwargs)	Invokes an *Operation* as a function.
`sole_output`()	

Gemm

class plaidml.op.**Gemm**(*a*, *b*, *c*, *alpha=None*, *beta=None*, *broadcast=True*, *transA=False*, *transB=False*)

Implements a general matrix multiplication.

Methods

`__init__`(a, b, c[, alpha, beta, broadcast, ...])	*Operation* constructor.
`bind`(bindings)	Builds an output variable dictionary for the operation.
`function`(*args, **kwargs)	Invokes an *Operation* as a function.
`sole_output`()	

Gradients

class plaidml.op.**Gradients**(*loss*, *variables*)

Compute the gradients of a loss with respect to a set of values

Methods

`__init__`(loss, variables)	*Operation* constructor.
`bind`(bindings)	Builds an output variable dictionary for the operation.
`function`(*args, **kwargs)	Invokes an *Operation* as a function.
`sole_output`()	

Hardmax

class plaidml.op.**Hardmax**(*data*)

Implements a standard ML hardmax.

Methods

`__init__`(data)	*Operation* constructor.
`bind`(bindings)	Builds an output variable dictionary for the operation.
`function`(*args, **kwargs)	Invokes an *Operation* as a function.
`sole_output`()	

Identity

class plaidml.op.**Identity**(*x*)
A simple identity operation.

Methods

__init__(x)	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

IsMax

class plaidml.op.**IsMax**(*value*, *axes*)
True iff an input's value is the maximum along some set of axes.

Methods

__init__(value, axes)	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

LogSoftmax

class plaidml.op.**LogSoftmax**(*data*)
Implements the log() of a standard ML softmax.

Methods

__init__(data)	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

MatMul

class plaidml.op.**MatMul**(*a*, *b*)
A matrix multiplication, using numpy semantics.

See https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.matmul.html for details.

Methods

__init__(a, b)	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

MaxPool

class plaidml.op.**MaxPool**(*data*, *padding*, *kernel_shape*, *pads*, *strides*)
A standard ML max pooling operator.

Methods

__init__(data, padding, kernel_shape, pads, ...)	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

MaxReduce

class plaidml.op.**MaxReduce**(*x*, *axes=None*, *keepdims=False*)
Computes the maximum value along some set of axes.

Methods

__init__(x[, axes, keepdims])	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

Mean

class plaidml.op.**Mean**(*x*, *axes=None*, *keepdims=False*, *floatx=<DType.FLOAT32: 50>*)
Computes the mean value along some set of axes.

Methods

__init__(x[, axes, keepdims, floatx])	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

MinReduce

class plaidml.op.**MinReduce**(*x*, *axes=None*, *keepdims=False*)
Computes the minimum value along some set of axes.

Methods

`__init__`(x[, axes, keepdims])	*Operation* constructor.
`bind`(bindings)	Builds an output variable dictionary for the operation.
`function`(*args, **kwargs)	Invokes an *Operation* as a function.
`sole_output`()	

NotEqual

class plaidml.op.**NotEqual**(*lhs*, *rhs*)

Elementwise tensor inequality.

Builds a boolean tensor whose values are true where the corresponding elements of the inputs are not equal.

Methods

`__init__`(lhs, rhs)	*Operation* constructor.
`bind`(bindings)	Builds an output variable dictionary for the operation.
`function`(*args, **kwargs)	Invokes an *Operation* as a function.
`sole_output`()	

Pow

class plaidml.op.**Pow**(*x*, *p*)

An elementwise pow() function.

Methods

`__init__`(x, p)	*Operation* constructor.
`bind`(bindings)	Builds an output variable dictionary for the operation.
`function`(*args, **kwargs)	Invokes an *Operation* as a function.
`sole_output`()	

Prod

class plaidml.op.**Prod**(*value*, *axes=None*, *keepdims=False*, *floatx=<DType.FLOAT32: 50>*)

Methods

`__init__`(value[, axes, keepdims, floatx])	*Operation* constructor.
`bind`(bindings)	Builds an output variable dictionary for the operation.
`function`(*args, **kwargs)	Invokes an *Operation* as a function.
`sole_output`()	

Relu

class plaidml.op.**Relu**(*x*, *alpha=None*, *max_value=None*)
A Rectified Linear Unit.

Methods

__init__(x[, alpha, max_value])	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

Reshape

class plaidml.op.**Reshape**(*x*, *dims*)
Reshapes a tensor, without changing the type or number of elements.

Methods

__init__(x, dims)	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

SliceTensor

class plaidml.op.**SliceTensor**(*data*, *axes=None*, *ends=None*, *starts=None*)
Implements tensor slicing.

Methods

__init__(data[, axes, ends, starts])	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

Softmax

class plaidml.op.**Softmax**(*data*)
Implements a standard ML softmax.

Methods

__init__(data)	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.

Continued on next page

Table 39 – continued from previous page

function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

Sqrt

class plaidml.op.**Sqrt**(*x*)

Computes the elementwise square root of a value.

Methods

__init__(x)	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

Summation

class plaidml.op.**Summation**(*value*, *axes=None*, *keepdims=False*, *floatx=<DType.FLOAT32: 50>*)

Sums an input value along some set of axes.

Methods

__init__(value[, axes, keepdims, floatx])	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

Variance

class plaidml.op.**Variance**(*x*, *axes=None*, *keepdims=False*, *floatx=<DType.FLOAT32: 50>*)

Methods

__init__(x[, axes, keepdims, floatx])	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

Functions

ceiling(data)	Elementwise ceiling.
clip(value, min_val, max_val)	
cos(data)	Elementwise cosine.
equal(lhs, rhs)	Elementwise tensor equality.

Continued on next page

Table 43 – continued from previous page

exp(data)	Elementwise exponential.
floor(data)	Elementwise floor.
gradients(loss, variables)	
hardmax(x[, axis])	
log(data)	Elementwise logarithm.
log_softmax(x[, axis])	
max_reduce(x[, axes, keepdims])	
mean(x[, axes, keepdims, floatx])	
min_reduce(x[, axes, keepdims])	
pad_compute(sym, input_size, filter_size, ...)	Computes info for an axis of a padded filter.
prod(value[, axes, keepdims, floatx])	
sigmoid(data)	Elementwise sigmoid.
sin(data)	Elementwise sine.
softmax(x[, axis])	
squeeze(x, axes)	
summation(value[, axes, keepdims, floatx])	
tanh(data)	Elementwise hyperbolic tangent.
unsqueeze(x, axes)	

ceiling

plaidml.op.**ceiling**(*data*)
Elementwise ceiling.

clip

plaidml.op.**clip**(*value*, *min_val*, *max_val*)

cos

plaidml.op.**cos**(*data*)
Elementwise cosine.

equal

plaidml.op.**equal**(*lhs*, *rhs*)
Elementwise tensor equality.

Builds a boolean tensor whose values are true when the corresponding elements of the inputs are equal.

Parameters

- **lhs** (tile.Value) – The left-hand side
- **rhs** (tile.Value) – The right-hand side

Returns *tile.Value* – The output value

exp

plaidml.op.**exp**(*data*)
Elementwise exponential.

floor

plaidml.op.**floor**(*data*)
Elementwise floor.

gradients

plaidml.op.**gradients**(*loss*, *variables*)

hardmax

plaidml.op.**hardmax**(*x*, *axis=None*)

log

plaidml.op.**log**(*data*)
Elementwise logarithm.

log_softmax

plaidml.op.**log_softmax**(*x*, *axis=None*)

max_reduce

plaidml.op.**max_reduce**(*x*, *axes=None*, *keepdims=False*)

mean

plaidml.op.**mean**(*x*, *axes=None*, *keepdims=False*, *floatx=<DType.FLOAT32: 50>*)

min_reduce

plaidml.op.**min_reduce**(*x*, *axes=None*, *keepdims=False*)

pad_compute

plaidml.op.**pad_compute**(*sym*, *input_size*, *filter_size*, *stride*, *padding*, *pads=None*)
Computes info for an axis of a padded filter.

Parameters

- **sym** (*str*) – The symbol for the input axis.
- **input_size** (*tile.Value or int*) – The size of the input axis (possibly symbolic).
- **filter_size** (*int*) – The size of the filter along this axis.
- **stride** (*int*) – The stride of the filter along this axis.
- **padding** (AutoPadding) – The padding style to use.
- **pads** (*(int, int) or None*) – Explicit pre- and post-padding for this axis.

Returns

tuple(A string representing the output size as TILE code, The pre-padding to use when building input accessor expressions, A tile.Value representing the computed output size)

prod

plaidml.op.**prod**(*value*, *axes=None*, *keepdims=False*, *floatx=<DType.FLOAT32: 50>*)

sigmoid

plaidml.op.**sigmoid**(*data*)
Elementwise sigmoid.

sin

plaidml.op.**sin**(*data*)
Elementwise sine.

softmax

plaidml.op.**softmax**(*x*, *axis=None*)

squeeze

plaidml.op.**squeeze**(*x*, *axes*)

summation

plaidml.op.**summation**(*value*, *axes=None*, *keepdims=False*, *floatx=<DType.FLOAT32: 50>*)

tanh

plaidml.op.**tanh**(*data*)
Elementwise hyperbolic tangent.

unsqueeze

plaidml.op.**unsqueeze**(*x*, *axes*)

5.3.3 plaidml.tile

Description

TILE program construction utilities.

When writing a program in TILE, you typically specify a graph of operations, where each operation is defined by a single TILE function. You then use composition to connect the individual TILE functions into a single composite

TILE function, which can then be compiled, scheduled, and executed as a single unit. The PlaidML API provides functions for building up and using these composite TILE programs.

Defining the individual per-operation TILE functions is sometimes trivial, and sometimes not. For example: although TILE makes it very easy to write a matrix multiply operation, the details of how that operations is expressed in TILE vary depending on the number of dimensions involved, whether broadcasting is required, &c. Higher-level frameworks tend to expect their backends to have a single "Do a matrix multiply" operation that's supposed to internally figure out all of these details; that's not really something that can be done in TILE directly.

It's wasteful and error-prone to implement these sometimes-tricky conversions from high-level semantics to low-level TILE code for each framework. And the frameworks tend to have similar semantics, thanks to the influence of Numpy. So PlaidML provides:

- A standard high-level operation library for constructing the per-operation TILE functions and composing them together (this module)
- Utilities to assist in constructing operations, such as broadcasting logic (this module)
- A suite of operations that we've found to be useful across a variety of frameworks (the *plaidml.op* module).

This library uses two passes for building up composite functions. The first pass constructs Python objects representing the operation graph; the second pass translates the operation graph to the composite TILE function. This is done for two reasons: it allows for higher-level optimizations (e.g. translating particular subtrees to more efficient TILE operations) and for expressing operations that cannot be efficiently implemented in the current version of TILE (e.g. it's very expensive to implement ArgMax in the initial released version of TILE, but ArgMax is typically used in composite expressions like `Equal(ArgMax(X), ArgMax(Y))`, which is trivial to efficiently implement in TILE).

More precisely, this library builds up a bipartite directed acyclic graph of `Operation` and `Value` objects. `Operation` is the base class of each operation; `Value` represents an operation input or output. `compose` translates an operation graph into a `plaidml.Function`.

See the Tile Tutorial for more information about how the TILE language works, or check out the PlaidML Op Tutorial for the details of writing your own operations.

Classes

DTypeInfo	Describes a PlaidML datatype.
Operation(code, inputs, outputs[, name, ...])	Operation base class.
Shape	Represents a symbolic tensor shape.
ShapeOf(x)	Computes the shape of a supplied tensor.
Source(op, output_name)	
Value(shape, var, source[, name])	A PlaidML variable and associated metadata.

DTypeInfo

class plaidml.tile.**DTypeInfo**

Describes a PlaidML datatype.

Methods

count(value)	
index(value, [start, [stop]])	Raises ValueError if the value is not present.

Attributes

base	Alias for field number 0
bitwidth	The number of bits occupied by an instance of the type.
width	Alias for field number 1

Operation

class plaidml.tile.**Operation**(*code*, *inputs*, *outputs*, *name=None*, *side_effects=None*)

Operation base class.

Methods

__init__(code, inputs, outputs[, name, ...])	*Operation* constructor.
bind(bindings)	Builds an output variable dictionary for the operation.
function(*args, **kwargs)	Invokes an *Operation* as a function.
sole_output()	

Shape

class plaidml.tile.**Shape**

Represents a symbolic tensor shape.

Methods

count(value)	
index(value, [start, [stop]])	Raises ValueError if the value is not present.

Attributes

dims	Alias for field number 1
dtype	Alias for field number 0
ndims	The shape's dimensionality.

ShapeOf

class plaidml.tile.**ShapeOf**(*x*)

Computes the shape of a supplied tensor.

(N.B. This is in tile.py instead of in op.py solely because it's useful to be able to reference it from Value.__getitem__(). For future-proofing, frameworks should reference this class as op.ShapeOf.)

Methods

`__init__`(x)	*Operation* constructor.
`bind`(bindings)	Builds an output variable dictionary for the operation.
`function`(*args, **kwargs)	Invokes an *Operation* as a function.
`sole_output`()	

Source

class plaidml.tile.**Source**(*op*, *output_name*)

Methods

`count`(value)	
`index`(value, [start, [stop]])	Raises ValueError if the value is not present.

Attributes

`op`	Alias for field number 0
`output_name`	Alias for field number 1

Value

class plaidml.tile.**Value**(*shape*, *var*, *source*, *name=None*)

A PlaidML variable and associated metadata.

Methods

`__init__`(shape, var, source[, name])	Constructs a *Value*.
`bind`(bindings)	Translates the *Value* to a PlaidML variable.
`for_op`(operation, output[, name])	Builds an operation output Value.
`from_dimensions`([dtype, name])	Builds an N-dimensional placeholder Value from a list of dimension sizes.
`from_ndims`([dtype, name])	Builds an N-dimensional placeholder Value.
`from_python_value`([dtype, name, ctx, dev])	Builds a Value from a Python value.
`from_value`()	
`from_var`(dimensions[, dtype, name])	Builds a Value from a PlaidML variable.
`is_bound`(bindings)	Indicates whether the *Value* has been bound.

Attributes

`key`	A dictionary key that unifies a Value with its shapeless slices.
`name`	

Functions

binary_op(lhs, rhs, op_str[, dtype, name])	Builds a Value for an elementwise binary operation.
broadcast_dims(*args)	Computes the broadcast dimensions of the supplied dimensions.
common_dtype(*args)	Finds the common dtype of a set of dtypes.
compose(ctx, dev, inputs, outputs[, updates])	Builds a TILE Function that computes the indicated values.
compute_aggregation_axes(dims[, axes, keepdims])	Computes parameters for an aggregation-over-axes operation.
maximum(x, y)	
minimum(x, y)	
to_dot(inputs, outputs[, updates, name])	Translates a chain of tensor computations to a DOT graph.
unary_op(value, op_str[, name])	Builds a Value for an elementwise unary operation.

binary_op

plaidml.tile.**binary_op**(*lhs*, *rhs*, *op_str*, *dtype=None*, *name=None*)

Builds a Value for an elementwise binary operation.

Parameters

- **lhs** (*Value or numeric*) – The left-hand side of the operation.
- **rhs** (*Value or numeric*) – The right-hand side of the operation.
- **op_str** (*str*) – The string to use for the operation. The string should be an expression in terms of 'L' and 'R'.
- **dtype** (*plaidml.DType*) – If not None, supplies the operation dtype; otherwise, the common dtype of lhs and rhs is used.
- **name** (*str*) – The name of the operation, or None.

Returns *Value* – A Value representing the result of the operation.

broadcast_dims

plaidml.tile.**broadcast_dims**(**args*)

Computes the broadcast dimensions of the supplied dimensions.

Parameters **args** (*[dims]*) – The list of dimension tuples to be broadcast.

Returns *tuple(Value)* – The broadcasted dims tuple.

common_dtype

plaidml.tile.**common_dtype**(**args*)

Finds the common dtype of a set of dtypes.

Parameters **args** (*[plaidml.DType]*) – The list of dtypes to be considered.

Returns *plaidml.DType* – The smallest dtype whose range encompasses the ranges of the supplied dtypes.

compose

plaidml.tile.**compose**(*ctx*, *dev*, *inputs*, *outputs*, *updates=None*)

Builds a TILE Function that computes the indicated values.

Parameters

- **ctx** (*plaidml.Context*) – The context to use for building the function.
- **dev** (plaidml.Device) – The device used to build the function (where constants will live)
- **inputs** (*[(name, Value)]*) – A list of named input placeholders.
- **outputs** (*[(name, Value)]*) – A list of named output values.
- **updates** (*[(original, updated)]*) – A list of updates to perform (side-effects).

Returns *plaidml.Invoker* – The composed TILE function.

compute_aggregation_axes

plaidml.tile.**compute_aggregation_axes**(*dims*, *axes=None*, *keepdims=False*)

Computes parameters for an aggregation-over-axes operation.

Parameters

- **dims** (*[int or Value]*) – The dimensions of the value being aggregated.
- **axes** (*[int], optional*) – Defaults to None. The indices of the axes to aggregate over.
- **keepdims** (*bool, optional*) – Defaults to False. Iff true, keep the aggregated axes in the output.

Returns

(dims, axes, dict(string->string)) –

The resulting dimensions and axes, and a dictionary of formatting replacements to use when building the TILE operation.

maximum

plaidml.tile.**maximum**(*x*, *y*)

minimum

plaidml.tile.**minimum**(*x*, *y*)

to_dot

plaidml.tile.**to_dot**(*inputs*, *outputs*, *updates=None*, *name='Tile'*)

Translates a chain of tensor computations to a DOT graph.

Parameters

- **inputs** (*[(name, Value)]*) – Provides names for computation inputs.
- **outputs** (*[(name, Value)]*) – The outputs to use for deriving the graph.

- **updates** (*[(original, updates)]*) – Side-effect updates that are part of the computations.
- **name** (*str*) – The name to use for the graph, or None.

Yields The strings comprising the lines of the DOT graph.

unary_op

plaidml.tile.**unary_op**(*value*, *op_str*, *name=None*)

Builds a Value for an elementwise unary operation.

Parameters

- **value** (Value) – The operation input.
- **op_str** (*str*) – The string to use for the operation. The string should be an expression in terms of 'I'.
- **name** (*str*) – The name of the operation, or None.

Returns *Value* – A Value representing the result of the operation.

Exceptions

Error	Errors raised during TILE function composition.
LogicError	Logic errors on the part of the caller.

Error

exception plaidml.tile.**Error**

Errors raised during TILE function composition.

LogicError

exception plaidml.tile.**LogicError**

Logic errors on the part of the caller.

5.3.4 plaidml.exceptions

Description

Exceptions

Aborted	A transactional operation was aborted by the system.
AlreadyExists	The requested object already exists.
Cancelled	Indicates that an asynchronous operations was cancelled.
DataLoss	The system has lost data required by the operation.
DeadlineExceeded	The operation deadline was exceeded.
FailedPrecondition	A precondition required by the operation is unmet.

Continued on next page

Table 57 – continued from previous page

Internal	An internal error occurred.
InvalidArgument	Indicates that at least one invalid argument was passed to a function.
NotFound	The requested object was not found.
OutOfRange	A call parameter is out of the range accepted by the implementation.
PermissionDenied	The caller does not have permission to access a required resource.
PlaidMLError	The base class of exceptions raised by VertexAI library operations.
ResourceExhausted	A resource required by the operation is exhausted.
Unauthenticated	The caller is unauthenticated.
Unavailable	A resource required by the operation is unavailable for use.
Unimplemented	The requested functionality is not implemented.
Unknown	A generic catch-all error.

Aborted

exception plaidml.exceptions.**Aborted**
A transactional operation was aborted by the system.

AlreadyExists

exception plaidml.exceptions.**AlreadyExists**
The requested object already exists.

Cancelled

exception plaidml.exceptions.**Cancelled**
Indicates that an asynchronous operations was cancelled.

DataLoss

exception plaidml.exceptions.**DataLoss**
The system has lost data required by the operation.

DeadlineExceeded

exception plaidml.exceptions.**DeadlineExceeded**
The operation deadline was exceeded.

FailedPrecondition

exception plaidml.exceptions.**FailedPrecondition**
A precondition required by the operation is unmet.

Internal

exception plaidml.exceptions.**Internal**
An internal error occurred.

InvalidArgument

exception plaidml.exceptions.**InvalidArgument**
Indicates that at least one invalid argument was passed to a function.

NotFound

exception plaidml.exceptions.**NotFound**
The requested object was not found.

OutOfRange

exception plaidml.exceptions.**OutOfRange**
A call parameter is out of the range accepted by the implementation.

PermissionDenied

exception plaidml.exceptions.**PermissionDenied**
The caller does not have permission to access a required resource.

PlaidMLError

exception plaidml.exceptions.**PlaidMLError**
The base class of exceptions raised by VertexAI library operations.

ResourceExhausted

exception plaidml.exceptions.**ResourceExhausted**
A resource required by the operation is exhausted.

Unauthenticated

exception plaidml.exceptions.**Unauthenticated**
The caller is unauthenticated.

Unavailable

exception plaidml.exceptions.**Unavailable**
A resource required by the operation is unavailable for use.

Unimplemented

exception plaidml.exceptions.**Unimplemented**
The requested functionality is not implemented.

Unknown

exception plaidml.exceptions.**Unknown**
A generic catch-all error.

5.4 Classes

Applier(ctx, f)	
Composer()	
context.Context(lib)	
DType	Describes the type of a tensor element.
Device(ctx, device)	
Dimension(size, stride)	
Function(code[, backtrace])	
Integer(value)	
Invocation(ctx, invoker)	
Invoker(ctx, f[, inputs, outputs])	
library.Library(lib[, logger])	A loaded PlaidML implementation library.
Placeholder(dims)	
Real(value)	
Shape(ctx, dtype, *args)	
Tensor(dev, shape[, copy_buffer])	
Var(v)	An abstract variable.

5.4.1 Applier

class plaidml.**Applier**(*ctx*, *f*)

Methods

__init__(ctx, f)	x.__init__(.
add_input(name, value)	
add_output(name)	
get_output_shape(name)	

5.4.2 Composer

class plaidml.**Composer**

Methods

`__init__`()	x.__init__(.
`add_dependency`(applier)	
`add_input`(name, val)	
`add_output`(name, val)	
`add_update`(dest, src)	
`build`()	

5.4.3 Context

class plaidml.context.**Context**(*lib*)

Methods

`__init__`(lib)	x.__init__(.
`cancel`()	
`set_eventlog_filename`(filename)	
`shutdown`()	

5.4.4 DType

class plaidml.**DType**

Describes the type of a tensor element.

Attributes

BOOLEAN
FLOAT16
FLOAT32
FLOAT64
INT16
INT32
INT64
INT8
INVALID
UINT16
UINT32
UINT64
UINT8

5.4.5 Device

class plaidml.**Device**(*ctx*, *device*)

Methods

__init__(ctx, device)	x.__init__(.
close()	
get_context()	

5.4.6 Dimension

class plaidml.**Dimension**(*size*, *stride*)

Methods

count(value)	
index(value, [start, [stop]])	Raises ValueError if the value is not present.

Attributes

size	Alias for field number 0
stride	Alias for field number 1

5.4.7 Function

class plaidml.**Function**(*code*, *backtrace=None*)

Methods

__init__(code[, backtrace])	x.__init__(.
save(filename)	

5.4.8 Integer

class plaidml.**Integer**(*value*)

Methods

__init__(value)	x.__init__(.

5.4.9 Invocation

class plaidml.**Invocation**(*ctx*, *invoker*)

Methods

__init__(ctx, invoker)	x.__init__(.

5.4.10 Invoker

class plaidml.**Invoker**(*ctx*, *f*, *inputs={}*, *outputs={}*)

Methods

__init__(ctx, f[, inputs, outputs])	x.__init__(.
get_output_shape(name)	
invoke()	
set_input(name, value)	
set_inputs(inputs)	
set_output(name, value)	
set_outputs(outputs)	

5.4.11 Library

class plaidml.library.**Library**(*lib*, *logger=<function log>*)

A loaded PlaidML implementation library.

Methods

__init__(lib[, logger])	x.__init__(.
get_perf_counter(name)	
last_status()	
raise_last_status()	
set_perf_counter(name, value)	

5.4.12 Placeholder

class plaidml.**Placeholder**(*dims*)

Methods

__init__(dims)	x.__init__(.

5.4.13 Real

class plaidml.**Real**(*value*)

Methods

__init__(value)	x.__init__(.

5.4.14 Shape

class plaidml.**Shape**(*ctx*, *dtype*, **args*)

Methods

__init__(ctx, dtype, *args)	x.__init__(.

Attributes

ctype
dimension_count
dimensions
dtype
offset
set_offset

5.4.15 Tensor

class plaidml.**Tensor**(*dev*, *shape*, *copy_buffer=False*)

Methods

__init__(dev, shape[, copy_buffer])	x.__init__(.
as_ndarray(ctx)	
mmap_current(**kwds)	
mmap_discard(**kwds)	

Attributes

buffer
shape

5.4.16 Var

class plaidml.**Var**(*v*)
An abstract variable.

Methods

`__init__`(v)	x.__init__(.

5.5 Functions

`plaidml.`**`devices`**(*ctx*, *limit=1*, *return_all=False*)
Returns a tuple of lists valid devices or aborts the program.

`plaidml.`**`get_perf_counter`**(*name*)

`plaidml.`**`gradients`**(*loss*, *variables*)

`plaidml.`**`load_function`**(*ctx*, *device*, *filename*)

`plaidml.`**`open_first_device`**(**args*, ***kwds*)

`plaidml.`**`quiet`**()

`plaidml.`**`run`**(*ctx*, *f*, *inputs={}*, *outputs={}*)

`plaidml.`**`set_backtrace`**(*enable*)

`plaidml.`**`set_floatx`**(*dtype*)

`plaidml.`**`set_perf_counter`**(*name*, *value*)

CHAPTER 6

Contributing to PlaidML

We welcome contributions to PlaidML from anyone. This document contains:

- Guidelines for creating successful PRs
- Outlines the contribution process
- Lists general areas for contribution
- Provides resources and context to ease development, where relevant and available

Before starting any work please ensure you are able to build and test PlaidML.

6.1 Guidelines

- Create unit tests for new features and bug fixes. Integration tests are required for larger features.
- Pre-commit linters will be available soon.
  - C++ code conforms to the Google C++ Style Guide.
  - Python code conforms to the Google Python Style Guide.

6.2 Process

1. Ensure there is an open issue assigned to you before doing (too much) work:
   - If you're tackling an open issue that isn't assigned, please assign it to yourself.
   - If you'd like to solve an issue that is already assigned, please comment on the issue to ensure you're not duplicating effort.
   - If you'd like to provide a new feature, open a new issue. Please provide a reasonably detailed description of what you'd like to do, and clearly indicate that you're willing to do the work.
2. Work on a fork as usual in Github. Please ensure the same tests travis runs will pass before creating a PR

3. Once you've created a PR you'll be asked to review and sign our Contributor License Agreement. You should review this before doing too much work to ensure the terms are acceptable.
4. Once tests have passed, a maintainer will assign the issue to themselves and run the PR through the (currently private) performance test suite. If there are issues, we will attempt to resolve them, but we may provide details and ask the author to address.
5. Once the performance regression suite has passed, we will accept and merge the PR.

6.3 Areas for Contribution

- Ops for Frontends
  - PlaidML welcomes implementations for currently unimplemented operations as well as Tile code for novel operations supported by research.
  - Please read the Tile Tutorial and the PlaidML Op Tutorial
- ML Framework Frontends (e.g., Keras, Pytorch, etc)
  - PlaidML welcomes integrations with any established ML framework or interop (NNVM, ONNX, etc)
  - Currently this involves duplicating tile operations. We will eventually abstract common NN tile operations into a separate C++ library to ease backend development.
- HALs for Backend Targets (OpenCL, Vulkan, SPIR-V, HVX, etc)
  - There is no documentation for the HAL currently. The interface is fairly straightforward and the OpenCL HAL provides a good example of a complete HAL.

Please follow the process above before embarking on anything major (like integrating a new frontend or backend).

CHAPTER 7

License

PlaidML is licensed under the AGPLv3.

Our open source goals include 1) helping students get started with deep learning as easily as possible and 2) helping researchers develop new methods more quickly than is possible with other tools. PlaidML is unique in being fully open source and free of dependence on libraries like cuDNN that carry revocable and redistribution-prohibiting licenses. For situations where an alternate license is preferable please contact solutions@vertex.ai.

CHAPTER 8

Reporting Issues

Either open a ticket on GitHub or post to plaidml-dev.

Python Module Index

p

Index