# TensorFlow

Курс “Практическое применение по TensorFlow”

Шигапова Фирюза Зинатуллаевна

1-й семестр, 2019 г.

https://github.com/Firyuza/TensorFlowPractice

# Custom C++ op

Reasons:

- There is no way to implement via existing TF and Python functions
- Existing operations are not performance or memory efficient

# Custom C++ op

How to:

- Register the new op in C++ file:
  - op’s name
  - input
  - output
  - define the shape function for tensor shape inference
- Concrete implementation of the op in C++
- Compile the op
- Load and use compiled op in TF program

# Custom C++ op. Register the new op

```
#include "tensorflow/core/framework/op.h"
#include "tensorflow/core/framework/shape_inference.h"
#include "tensorflow/core/framework/op_kernel.h"

using namespace tensorflow;

REGISTER_OP("ZeroOut")
    .Input("to_zero: int32")
    .Output("zeroed: int32")
    .SetShapeFn([](::tensorflow::shape_inference::InferenceContext* c) {
      c->set_output(0, c->input(0));
      return Status::OK();
    });
```

# Custom C++ op. Implement kernel for the op

```
class ZeroOutOp : public OpKernel {
 public:
  explicit ZeroOutOp(OpKernelConstruction* context) : OpKernel(context) {}

  void Compute(OpKernelContext* context) override {
    // Grab the input tensor
    const Tensor& input_tensor = context->input(0);
    auto input = input_tensor.flat<int32>();

    // Create an output tensor
    Tensor* output_tensor = NULL;
    OP_REQUIRES_OK(context, context->allocate_output(0, input_tensor.shape(),
                                                     &output_tensor));
    auto output_flat = output_tensor->flat<int32>();

    // Set all but the first element of the output tensor to 0.
    const int N = input.size();
    for (int i = 1; i < N; i++) {
      output_flat(i) = 0;
    }

    // Preserve the first input value if possible.
    if (N > 0) output_flat(0) = input(0);
  }
};
```

Extend OpKernel

Override Compute method

Access the input and output tensors

# Custom C++ op. Define constraints for kernel run

- CPU kernel

```
REGISTER_KERNEL_BUILDER(Name("ZeroOut").Device(DEVICE_CPU), ZeroOutOp);
```

- Multi-threaded CPU kernels
- GPU kernels (include CUDA code)

# Custom C++ op. Compile op

In terminal run:

1. Activate your environment
2. `TF_CFLAGS=( $(python -c 'import tensorflow as tf; print(" ".join(tf.sysconfig.get_compile_flags()))') )`
3. `TF_LFLAGS=( $(python -c 'import tensorflow as tf; print(" ".join(tf.sysconfig.get_link_flags()))') )`

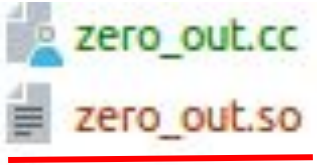
# Custom C++ op. Compile op

4. `g++ -std=c++11 -shared zero_out.cc -o zero_out.so -fPIC ${TF_CFLAGS[@]} ${TF_LFLAGS[@]} -O2`

5. If gcc >=5 add to the 4 step **`-D_GLIBCXX_USE_CXX11_ABI=0`**

# Custom C++ op. Compile op

After all



# Custom C++ op. Load

Load Op by

```
zero_out_module = tf.load_op_library('./zero_out.so')
```

# Custom C++ op. Test it

```
import tensorflow as tf

class ZeroOutTest(tf.test.TestCase):
  def testZeroOut(self):
    zero_out_module = tf.load_op_library('./zero_out.so')
    with self.test_session():
      result = zero_out_module.zero_out([5, 4, 3, 2, 1])
      self.assertAllEqual(result.numpy(), [5, 0, 0, 0, 0])

if __name__ == "__main__":
  tf.test.main()
```

!!!**snake_case** name
(in C++ it's ZeroOut => in Python it's zero_out