Developing New Layers in Caffe

Kai KANG
kkang@ee.cuhk.edu.hk

Outline

- Look before You Leap
- Dive into Development
- Test Your Layers

Look Before You Leap

Look Before You Jump

- Do you really need a new layer?
- Have others implemented it yet?
- Is it worth the effort?

Do You Need a New Layer?

- Caffe may already have the layer
  - Read the documentations (http://caffe.berkeleyvision.org/doxygen/annotated.html)
- You may combine existing layers to have same effect
- Discover special usages of existing layers to simply the problem

Example - Sum Feature Maps

- Use 1x1 convolutions
  - Weights initialized to 1
  - Bias initialized to 0
  - Learning rates and weight decays set to 0

```
layer {
  name: "fusion"
  type: "Convolution"
  bottom: “feature_maps"
  top: "fusion"
  param { lr_mult: 0 decay_mult: 0 }
  param { lr_mult: 0 decay_mult: 0 }
  convolution_param {
    num_output: 1
    pad: 0
    kernel_size: 1
    stride: 1
    weight_filler { type: “constant" value: 1 }
    bias_filler { type: “constant" value: 0 }
  }
}
```

Example - Binary Error

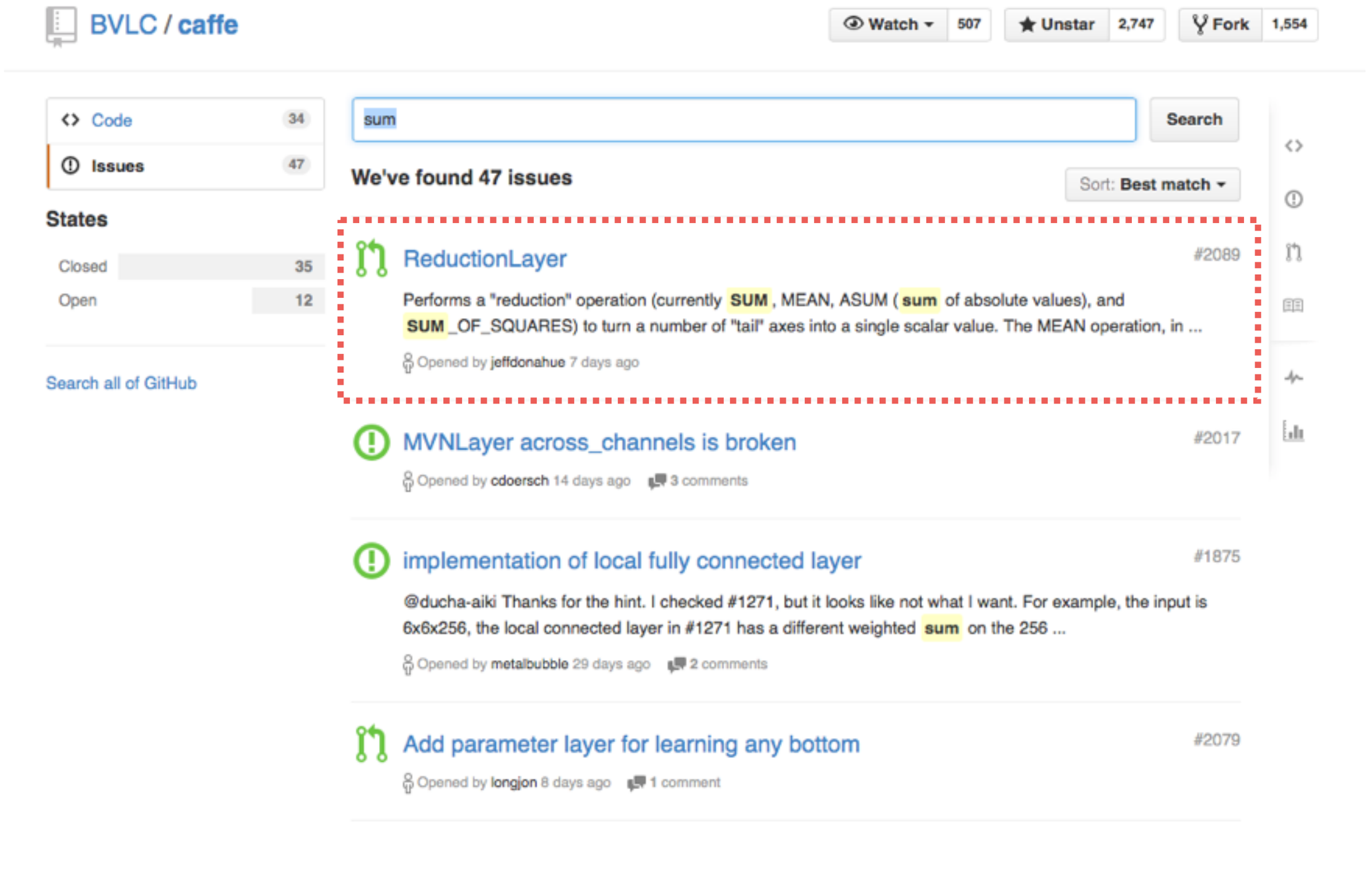
- Threshold layer to get decisions
- Elementwise layer to calculate difference
- Absolute layer to get error
- Write new layer to sum all elements

```
layer {
  name: "fusion"
  type: "Convolution"
  bottom: “feature_maps"
  top: "fusion"
  param { lr_mult: 0 decay_mult: 0 }
  param { lr_mult: 0 decay_mult: 0 }
  convolution_param {
    num_output: 1
    pad: 0
    kernel_size: 1
    stride: 1
    weight_filler { type: “constant" value: 1 }
    bias_filler { type: “constant" value: 0 }
  }
}
```

Have Others Done It?

- Discuss with other users
  - Colleagues or labmates
  - Caffe-user group
- Search the web
  - Search Issues or Pull-request on GitHub
  - Search related papers and ask for code
  - Search other users repositories

Example - Sum All Elements



Example - Sum All Elements

ReductionLayer #2089

Open **jeffdonahue** wants to merge 1 commit into `BVLC:master` from `jeffdonahue:reduction-layer`

Change Details

Conversation 0 · Commits 1 · Files changed 5 +508 −1

jeffdonahue commented 7 days ago — Owner

Performs a "reduction" operation (currently SUM, MEAN, ASUM (sum of absolute values), and SUM_OF_SQUARES) to turn a number of "tail" axes into a single scalar value. The `MEAN` operation, in combination with a `loss_weight`, is useful for creating custom losses that don't have an obnoxious amount of output. For example, this `EuclideanLoss`:

```
layer {
  type: "EuclideanLoss"
  bottom: "preds"
  bottom: "targets"
  top: "mean_squared_error"
  loss_weight: 1
}
```

...is equivalent to this `Reduction`:

```
layer {
  type: "Eltwise"
  bottom: "preds"
  bottom: "targets"
```

Labels: ready for review

Milestone: No milestone

Assignee: No one assigned

Notifications: Unsubscribe

You're receiving notifications because you're subscribed to this thread.

1 participant

Subscribe to get updates

Is It Worth the Effort?

- Writing new layers takes time, especially for the first time
- Maybe other frameworks are better options
  - Unsupervised training
  - Generative models

Dive into Development

Dive into Development

- Choose a good starting point
- Find a reference layer
- Follow the development procedure

Starting Point

- Clone the latest Caffe repository from GitHub
  - Avoid known issues
    - Newer versions make development easier
    - Easier to maintain for the future
- Compile and run the test first
- Checkout a new branch and start writing

```
# Clone the latest repository from GitHub
# https://github.com/BVLC/caffe
git clone git@github.com:BVLC/caffe.git

# Compile and run the test
# http://caffe.berkeleyvision.org/installation.html
make all
make test
make runtest

# Checkout a new branch and start writing
git checkout -b mydev
```

Reference Layer

- Layer types
  - Common layers: ArgMax, ConCat, Eltwise, Flatten, InnerProduct, Silence, Softmax…
  - Data layers: Data, ImageData, WindowData…
  - Loss layers: Accuracy, EuclideanLoss, HingeLoss, SigmoidCrossEntropy…
  - Vision layers: Convolution, Pooling, Deconvolution
  - Neuron layers: ReLU, Sigmoid, Tanh…

Development Procedure

- Development wiki (https://github.com/BVLC/caffe/wiki/Development) for latest guides
- Add class declarations
- Implement you layers in C++ and CUDA
- Add Protocol Buffers parameters
- Add layer instantiation and registration
- Compile and debug

1. Add Class Declaration

- Add a class declaration in one of **`common_layer.hpp`**, **`data_layers.hpp`**, **`loss_layers.hpp`**, **`neuron_layers.hpp`** or **`vision_layers.hpp`**
  - Include an inline implementation of **`type()`** and **`*Blobs()`** methods
  - Omit **`*_gpu`** declarations if you'll only implement CPU code

Example - MapDataLayer

data_layer.hpp

```
template<typename Dtype>
class MapDataLayer : public BasePrefetchingDataLayer<Dtype> {
 public:
  explicit MapDataLayer(const LayerParameter& param)
      : BasePrefetchingDataLayer<Dtype>(param),
        label_transformer_(label_trans_param(param.transform_param()), this->phase_) {}
  virtual ~MapDataLayer();
  virtual void DataLayerSetUp(const vector<Blob<Dtype>*>& bottom,
      const vector<Blob<Dtype>*>& top);

  virtual inline const char* type() const { return "MapData"; }
  virtual inline int ExactNumBottomBlobs() const { return 0; }
  virtual inline int ExactNumTopBlobs() const { return 2; }

 protected:
  virtual void InternalThreadEntry();
  DataTransformer<Dtype> label_transformer_;
  Blob<Dtype> transformed_label_;

  shared_ptr<db::DB> db_;
  shared_ptr<db::Cursor> iter_;

 private:
  static TransformationParameter label_trans_param(
            const TransformationParameter& trans_param);
};
```

type()

*Blobs()

2. Implementation

- Implement your layer in layers/your_layer.cpp
  - (optional) LayerSetUp method for one-time initialization
  - Reshape method for computing sizes of top blobs, allocating buffers
  - Forward_cpu for forward propagation
  - Backward_cpu for gradient computations if needed
- Implement GPU versions Forward_gpu and Backward_gpu in layers/your_layer.cu

Example - MapDataLayer

map_data_layer.cpp

```
template <typename Dtype>
void MapDataLayer<Dtype>::DataLayerSetUp(const vector<Blob<Dtype>*>& bottom,
      const vector<Blob<Dtype>*>& top) {
  // Initialize DB
  // Read a data point and use it to initialize the top blob.
  // reshape data map
}

template<typename Dtype>
void MapDataLayer<Dtype>::InternalThreadEntry() {
  // Variable declarations
  // ...

  for (int item_id = 0; item_id < batch_size; ++item_id) {
    maps.ParseFromString(iter_->value());
    // Apply data and label transformations (mirror, scale, crop...)

    // go to the next iter
    iter_->Next();
    if (!iter_->valid()) {
      iter_->SeekToFirst();
    }
  }
}
```

Forward() for PrefetchDataLayer

3. Proto Declarations

- If needed, declare parameters in proto/caffe.proto

```
// NOTE
// Update the next available ID when you add a new LayerParameter field.
//
// LayerParameter next available layer-specific ID: 135 (last added:
reduction_param)
message LayerParameter {
  …
  // Layer type-specific parameters.
  //
  // Note: certain layers may have more than one computational engine
  // for their implementation. These layers include an Engine type and
  // engine parameter for selecting the implementation.
  // The default for the engine is set by the ENGINE switch at compile-
time.
  optional AccuracyParameter accuracy_param = 102;
  optional ArgMaxParameter argmax_param = 103;
  optional ConcatParameter concat_param = 104;
  optional MyLayerParameter mylayer_param = 135;
…
}
```

3. Proto Declarations

- If needed, declare parameters in proto/caffe.proto

```
// Message that stores parameters used by EltwiseLayer
message EltwiseParameter {
  enum EltwiseOp {
    PROD = 0;
    SUM = 1;
    MAX = 2;
  }
  optional EltwiseOp operation = 1 [default = SUM]; // element-wise
operation
  repeated float coeff = 2; // blob-wise coefficient for SUM operation

  // Whether to use an asymptotically slower (for >2 inputs) but stabler
method
  // of computing the gradient for the PROD operation. (No effect for SUM
op.)
  optional bool stable_prod_grad = 3 [default = true];
}
```

3. Layer Instantiation and Registration

- Add layer instantiation and registration in **`my_layer.cpp`**

```
INSTANTIATE_CLASS(MyLayer);
REGISTER_LAYER_CLASS(My);
```

- And in my_layer.cu

```
INSTANTIATE_LAYER_GPU_FUNCS(MyLayer);
```

Layer Testing and Maintenance

Layer Testing and Maintenance

- Write tests in `test/test_your_layer.cpp`.
- Use `test/test_gradient_check_util.hpp` to check Forward and Backward implementations are in numerical agreement
- Write tests in different scenarios
- Compile, run tests and fix bugs
- Frequently pull the latest changes and update your own layers if there are conflicts