Caffe

Deep learning framework by the BVLC

Created by
Yangqing Jia
Lead Developer
Evan Shelhamer

View On GitHub

Siamese Network Training with Caffe

This example shows how you can use weight sharing and a contrastive loss function to learn a model using a siamese network in Caffe.

We will assume that you have caffe successfully compiled. If not, please refer to the Installation page. This example builds on the MNIST tutorial so it would be a good idea to read that before continuing.

The guide specifies all paths and assumes all commands are executed from the root caffe directory

Prepare Datasets

You will first need to download and convert the data from the MNIST website. To do this, simply run the following commands:

```
./data/mnist/get_mnist.sh
./examples/siamese/create_mnist_siamese.sh
```

After running the script there should be two datasets,

`./examples/siamese/mnist_siamese_train_leveldb`, and

`./examples/siamese/mnist_siamese_test_leveldb`.

The Model

First, we will define the model that we want to train using the siamese network. We will use the convolutional net defined in `./examples/siamese/mnist_siamese.prototxt`. This model is almost exactly the same as the LeNet model, the only difference is that we have replaced the top layers that produced probabilities over the 10 digit classes with a linear "feature" layer that produces a 2 dimensional vector.

```
layer {
  name: "feat"
  type: "InnerProduct"
  bottom: "ip2"
  top: "feat"
  param {
    name: "feat_w"
    lr_mult: 1
  }
  param {
    name: "feat_b"
    lr_mult: 2
  }
  inner_product_param {
    num_output: 2
  }
}
```

Define the Siamese Network

In this section we will define the siamese network used for training. The resulting network is defined in `./examples/siamese/mnist_siamese_train_test.prototxt`.

Reading in the Pair Data

We start with a data layer that reads from the LevelDB database we created earlier. Each entry in this database contains the image data for a pair of images (`pair_data`) and a binary label saying if they belong to the same class or different classes (`sim`).

```
layer {
  name: "pair_data"
  type: "Data"
  top: "pair_data"
  top: "sim"
  include { phase: TRAIN }
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "examples/siamese/mnist_siamese_train_leveldb"
    batch_size: 64
  }
```

```
}
```

In order to pack a pair of images into the same blob in the database we pack one image per channel. We want to be able to work with these two images separately, so we add a slice layer after the data layer. This takes the `pair_data` and slices it along the channel dimension so that we have a single image in `data` and its paired image in `data_p`.

```
layer {
  name: "slice_pair"
  type: "Slice"
  bottom: "pair_data"
  top: "data"
  top: "data_p"
  slice_param {
    slice_dim: 1
    slice_point: 1
  }
}
```

Building the First Side of the Siamese Net

Now we can specify the first side of the siamese net. This side operates on `data` and produces `feat`. Starting from the net in `./examples/siamese/mnist_siamese.prototxt` we add default weight fillers. Then we name the parameters of the convolutional and inner product layers. Naming the parameters allows Caffe to share the parameters between layers on both sides of the siamese net. In the definition this looks like:

```
...
param { name: "conv1_w" ...  }
param { name: "conv1_b" ...  }
...
param { name: "conv2_w" ...  }
param { name: "conv2_b" ...  }
...
param { name: "ip1_w" ...  }
param { name: "ip1_b" ...  }
...
param { name: "ip2_w" ...  }
param { name: "ip2_b" ...  }
...
```

Building the Second Side of the Siamese Net

Now we need to create the second path that operates on `data_p` and produces `feat_p`. This path is exactly the same as the first. So we can just copy and paste it. Then we change the name of each layer, input, and output by appending `_p` to differentiate the "paired" layers from the originals.

Adding the Contrastive Loss Function

To train the network we will optimize a contrastive loss function proposed in: Raia Hadsell, Sumit Chopra, and Yann LeCun "Dimensionality Reduction by Learning an Invariant Mapping". This loss function encourages matching pairs to be close together in feature space while pushing non-matching pairs apart. This cost function is implemented with the `CONTRASTIVE_LOSS` layer:

```
layer {
    name: "loss"
    type: "ContrastiveLoss"
    contrastive_loss_param {
        margin: 1.0
    }
    bottom: "feat"
    bottom: "feat_p"
    bottom: "sim"
    top: "loss"
}
```

Define the Solver

Nothing special needs to be done to the solver besides pointing it at the correct model file. The solver is defined in `./examples/siamese/mnist_siamese_solver.prototxt`.

Training and Testing the Model

Training the model is simple after you have written the network definition protobuf and solver protobuf files. Simply run `./examples/siamese/train_mnist_siamese.sh`:

```
./examples/siamese/train_mnist_siamese.sh
```

Plotting the results

First, we can draw the model and siamese networks by running the following commands that draw the DAGs defined in the .prototxt files:

```
./python/draw_net.py \
    ./examples/siamese/mnist_siamese.prototxt \
    ./examples/siamese/mnist_siamese.png

./python/draw_net.py \
    ./examples/siamese/mnist_siamese_train_test.prototxt \
    ./examples/siamese/mnist_siamese_train_test.png
```

Second, we can load the learned model and plot the features using the iPython notebook:

```
ipython notebook ./examples/siamese/mnist_siamese.ipynb
```