Applied Parallel Computing

parallel-computing.pro

CAFFE framework

Layers

- Data layers
  - Image Data - read raw images.
  - Database - read data from LEVELDB or LMDB.
  - HDF5 Input - read HDF5 data, allows data of arbitrary dimensions.
  - HDF5 Output - write data as HDF5.
  - Input - typically used for networks that are being deployed.
  - Window Data - read window data file.
  - Memory Data - read data directly from memory.
  - Dummy Data - for static data and debugging.

Layers

- Vision layers
  - Convolution Layer - convolves the input image with a set of learnable filters, each producing one feature map in the output image.
  - Pooling Layer - max, average, or stochastic pooling.
  - Spatial Pyramid Pooling (SPP)
  - Crop - perform cropping transformation.
  - Deconvolution Layer - transposed convolution.

Layers

- Recurrent layers
  - Recurrent
  - RNN
  - Long-Short Term Memory (LSTM)

Layers

- Common layers
  - Inner Product - fully connected layer.
  - Dropout
  - Embed - for learning embeddings of one-hot encoded vector (takes index as input).

Layers

- Normalization layers
  - Local Response Normalization (LRN) - performs a kind of “lateral inhibition” by normalizing over local input regions.
  - Mean Variance Normalization (MVN) - performs contrast normalization / instance normalization.
  - Batch Normalization - performs normalization over mini-batches.
  - The bias and scale layers can be helpful in combination with normalization.

Layers

- Activation / Neuro layers
  - ReLU / Rectified-Linear and Leaky-ReLU - ReLU and Leaky-ReLU rectification.
  - PReLU - parametric ReLU.
  - ELU - exponential linear rectification.
  - Sigmoid
  - TanH
  - Absolute Value
  - Power - f(x) = (shift + scale * x) ^ power.
  - Exp - f(x) = base ^ (shift + scale * x).
  - Log - f(x) = log(x).
  - BNLL - f(x) = log(1 + exp(x)).
  - Threshold - performs step function at user defined threshold.
  - Bias - adds a bias to a blob that can either be learned or fixed.
  - Scale - scales a blob by an amount that can either be learned or fixed.

Layers

- Utility layers
  - Flatten
  - Reshape
  - Batch Reindex
  - Split
  - Concat
  - Slicing
  - Eltwise - element-wise operations such as product or sum between two blobs.
  - Filter / Mask - mask or select output using last blob.
  - Parameter - enable parameters to be shared between layers.
  - Reduction - reduce input blob to scalar blob using operations such as sum or mean.
  - Silence - prevent top-level blobs from being printed during training.
  - ArgMax
  - Softmax
  - Python - allows custom Python layers.

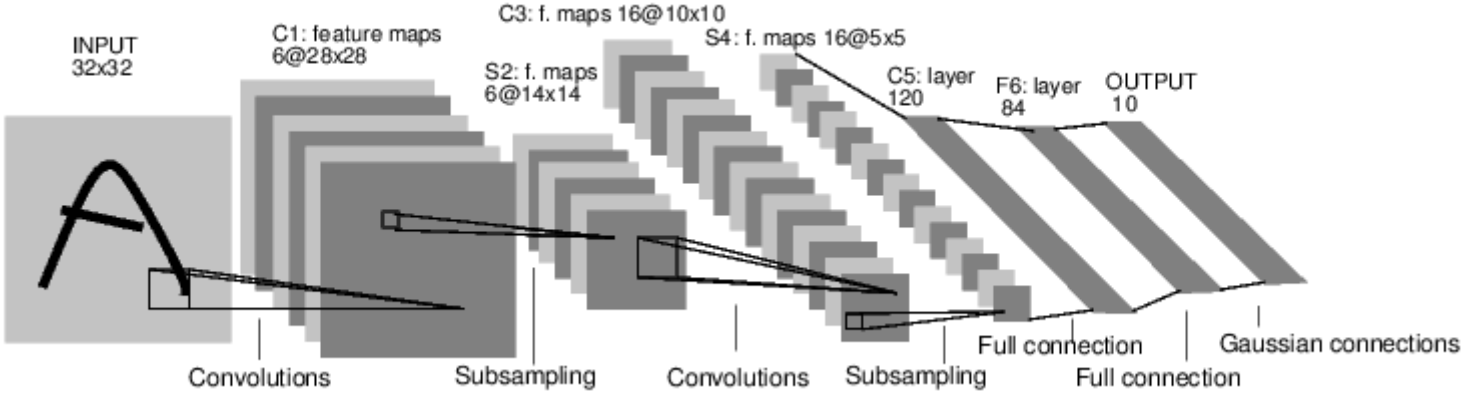
Layers

- Common layers
  - Multinomial Logistic Loss
  - Infogain Loss - a generalization of Multinomial Logistic Loss Layer.
  - Softmax with Loss - computes the multinomial logistic loss of the softmax of its inputs. It's conceptually identical to a softmax layer followed by a multinomial logistic loss layer, but provides a more numerically stable gradient.
  - Sum-of-Squares / Euclidean - computes the sum of squares of differences of its two inputs,
  - Hinge / Margin - The hinge loss layer computes a one-vs-all hinge (L1) or squared hinge loss (L2).
  - Sigmoid Cross-Entropy Loss - computes the cross-entropy (logistic) loss, often used for predicting targets interpreted as probabilities.
  - Accuracy / Top-k layer - scores the output as an accuracy with respect to target – it is not actually a loss and has no backward step.
  - Contrastive Loss

MNIST

```
cd $CAFFE_ROOT
./data/mnist/get_mnist.sh
./examples/mnist/create_mnist.sh
```

LeNet model


INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

Data load layer

```
layer {
  name: "mnist"
  type: "Data"
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "mnist_train_lmdb"
    backend: LMDB
    batch_size: 64
  }
  top: "data"
  top: "label"
}
```

Convolution layer

```
layer {
  name: "conv1"
  type: "Convolution"
  param { lr_mult: 1 }
  param { lr_mult: 2 }
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
  bottom: "data"
  top: "conv1"
}
```

Pooling layer

```
layer {
  name: "pool1"
  type: "Pooling"
  pooling_param {
    kernel_size: 2
    stride: 2
    pool: MAX
  }
  bottom: "conv1"
  top: "pool1"
}
```

Fully connected layer

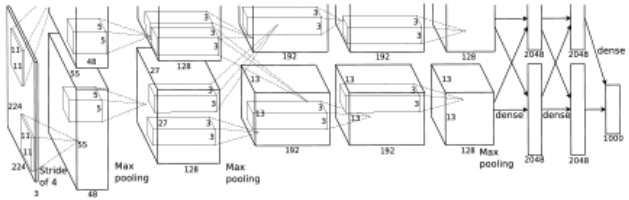
```
layer {
  name: "ip1"
  type: "InnerProduct"
  param { lr_mult: 1 }
  param { lr_mult: 2 }
  inner_product_param {
    num_output: 500
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
  bottom: "pool2"
  top: "ip1"
}
```

Blobs, layers, nets

- Caffe stores, communicates, and manipulates the information as *blobs*: the blob is the standard array and unified memory interface for the framework.
- The layer comes next as the foundation of both model and computation.
- The net follows as the collection and connection of layers.
- The details of blob describe how information is stored and communicated in and across layers and nets.

Forward/Backward

Forward: inference $f_W(x)$

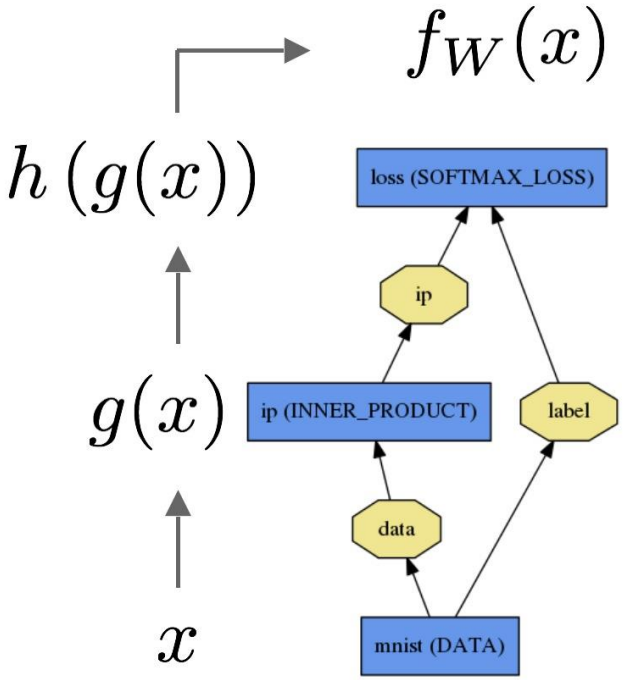


"espresso" + loss

$\nabla f_W(x)$ Backward: learning

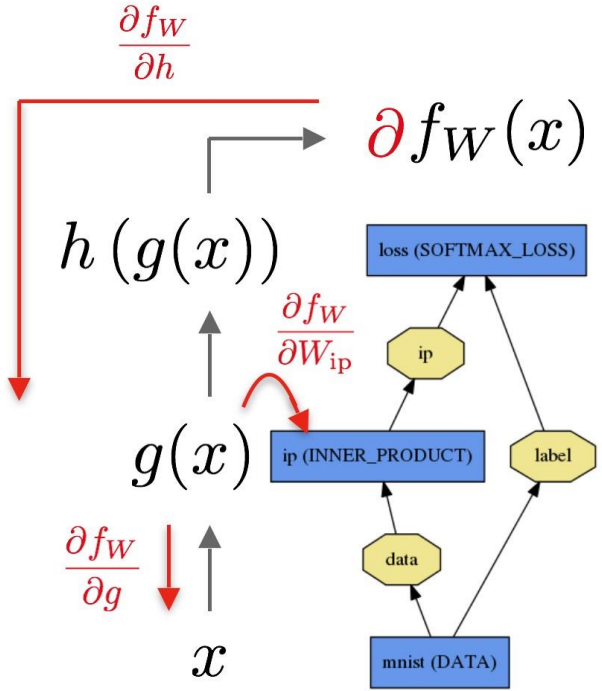
Forward pass

- The **forward** pass computes the output given the input for inference.
- In forward Caffe composes the computation of each layer to compute the “function” represented by the model. This pass goes from bottom to top.



Backward pass

- The **backward** pass computes the gradient given the loss for learning.
- In backward Caffe reverse-composes the gradient of each layer to compute the gradient of the whole model by automatic differentiation.
- This is back-propagation. This pass goes from top to bottom.



Caffe models

- A caffe model is distributed as a directory containing:
  - Solver/model prototxt(s)
  - readme.md containing
    - ✓ YAML frontmatter
      - Caffe version used to train this model (tagged release or commit hash).
      - [optional] file URL and SHA1 of the trained .caffemodel.
      - [optional] github gist id.
    - ✓ Information about what data the model was trained on, modeling choices, etc.
    - ✓ License information.
  - [optional] Other helpful scripts.
- https://github.com/BVLC/caffe/wiki/Model-Zoo