Caffe

Deep learning framework by the BVLC

Created by
Yangqing Jia
Lead Developer
Evan Shelhamer

View On GitHub

Loss

In Caffe, as in most of machine learning, learning is driven by a **loss** function (also known as an **error**, **cost**, or **objective** function). A loss function specifies the goal of learning by mapping parameter settings (i.e., the current network weights) to a scalar value specifying the "badness" of these parameter settings. Hence, the goal of learning is to find a setting of the weights that *minimizes* the loss function.

The loss in Caffe is computed by the Forward pass of the network. Each layer takes a set of input (`bottom`) blobs and produces a set of output (`top`) blobs. Some of these layers' outputs may be used in the loss function. A typical choice of loss function for one-versus-all classification tasks is the `SoftmaxWithLoss` function, used in a network definition as follows, for example:

```
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "pred"
  bottom: "label"
  top: "loss"
}
```

In a `SoftmaxWithLoss` function, the `top` blob is a scalar (empty shape) which

averages the loss (computed from predicted labels `pred` and actuals labels `label`) over the entire mini-batch.

Loss weights

For nets with multiple layers producing a loss (e.g., a network that both classifies the input using a `SoftmaxWithLoss` layer and reconstructs it using a `EuclideanLoss` layer), *loss weights* can be used to specify their relative importance.

By convention, Caffe layer types with the suffix `Loss` contribute to the loss function, but other layers are assumed to be purely used for intermediate computations. However, any layer can be used as a loss by adding a field `loss_weight: <float>` to a layer definition for each `top` blob produced by the layer. Layers with the suffix `Loss` have an implicit `loss_weight: 1` for the first `top` blob (and `loss_weight: 0` for any additional `top`s); other layers have an implicit `loss_weight: 0` for all `top`s. So, the above `SoftmaxWithLoss` layer could be equivalently written as:

```
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "pred"
  bottom: "label"
  top: "loss"
  loss_weight: 1
}
```

However, *any* layer able to backpropagate may be given a non-zero `loss_weight`, allowing one to, for example, regularize the activations produced by some intermediate layer(s) of the network if desired. For non-singleton outputs with an associated non-zero loss, the loss is computed simply by summing over all entries of the blob.

The final loss in Caffe, then, is computed by summing the total weighted loss over the network, as in the following pseudo-code:

```
loss := 0
for layer in layers:
  for top, loss_weight in layer.tops, layer.loss_weights:
    loss += loss_weight * sum(top)
```