Caffe – Deep Learning Framework

- SHUBHAM RANDER
Do we need a Deep Learning Framework?

Before answering this question, let’s look into inner workings of Neural Networks.
Simple Neural Network

Linear model: \( y = w \cdot x + b \)

\[ y = g(w \cdot x + b) \]

\[ y = g(Wx + b) \]

Nonlinear transformation, Warp space, Shift

Credits: https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/
Simple Neural Network

Linear Model

Using Neural Network

Possible because of Non Linear space transformation

Credits: https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/
Deep Neural Network

\[ y = g(Wx + b) \]
\[ z = g(Vy + c) \]
\[ z = g(Vg(Wx + b) + c) \]

output of first layer
Space Warping using NN

Credits: https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/
Classification using Neural Network

\[ P(y|x) = \text{softmax}(Wg(Vf(x))) \]

**Softmax**

\[ P(y|x) = \frac{\exp(w^T f(x, y))}{\sum_{y'} \exp(w^T f(x, y'))} \]
Training Neural Networks

• Maximum Log Likelihood of Training Data

\[ \mathcal{L}(x, i^*) = \log P(y = i^* | x) = \log \left( \text{softmax}(Wz) \cdot e_{i^*} \right) \]

\[ \mathcal{L}(x, i^*) = Wz \cdot e_{i^*} - \log \sum_j \exp(Wz) \cdot e_j \]
Computing Gradient

\[ \frac{\partial}{\partial W_{ij}} L(x, i^*) = \begin{cases} z_j - P(y = i|x)z_j & \text{if } i = i^* \\ -P(y = i|x)z_j & \text{otherwise} \end{cases} \]
Backpropagation

- Now once the gradients for $W$ have been calculated now we have to calculate the gradients of other weights that is for $V$.

\[
\frac{\partial L(x, i^*)}{\partial V_{ij}} = \frac{\partial L(x, i^*)}{\partial z} \frac{\partial z}{\partial V_{ij}}
\]

[some math...]

\[
err(\text{root}) = e_{i^*} - P(y|x)
\]

\[
dim = m
\]

\[
\frac{\partial L(x, i^*)}{\partial z} = err(z) = W^T err(\text{root})
\]

\[
dim = d
\]
Backpropagation

Can forget everything after z, treat it as the output and keep backproping

Credits: 3Blue1Brown Youtube Video Series
Imagine complexity of Forward Propagation and Backpropagation in deep networks

Inception GoogLeNet

- 22 Layer Deep Architecture
- ~4 million parameters to train
Imagine complexity of Forward Propagation and Backpropagation in deep networks

ResNet
So...

- Things get more complex with addition of deep learning features like:
  - Batch Normalization
  - Pooling
  - Convolutional layers
  - Dilation in convoluted layers
  - Recurrent Layers etc.

- Building computation graphs for both forward and backward propagation becomes very cumbersome

- Same code should support running on both GPU and CPU, or else we may have to write different versions of same code.
So...

- If we need to change the activation layer of one layer, we may have to change a lot of code in computation of both forward and backward propagation
- Hyper Parameter tuning becomes more challenging
- Plugging in or removing extra layers will not be straightforward
- Sometimes should be compatible on running different platforms like MATLAB and Python
- Compiler Optimizations
- ... many more
The point of Deep Learning Frameworks

- Build big computational graphs easily
- Compute gradients in computational graphs easily
- Run it all efficiently on GPU and CPU (wrap cuDNN, cuBLAS, etc)
How does it help Engineer/Scientist

- Allows them to focus on problem by abstracting the intricate math details
- Allows faster prototyping of models in local and enable them to run cloud clusters/high performance clusters seamlessly
- Easier experimentation with different layers and hyper parameter tuning
Finally let’s come to Caffe

- Caffe stands for - Convolutional Architecture for Fast Feature Embedding

- Created by Yangqing Jia at Berkeley AI Research (BAIR)

- It is open source

- Primarily designed to solve problems in the field for Computer Vision, later adopted in the field of speech recognition, robotics, neuroscience, and astronomy
What is in for us?

- Caffe provides a complete toolkit for training, testing, fine tuning, and deploying models, with well-documented examples for all of these tasks.

- Provides state-of-the-art deep learning models (AlexNet) for reference

- Provided fastest available implementations at the time
Philosophy behind developing Caffe

- Expression
- Speed
- Modularity
- Openness
- Community
Philosophy: Expression

Models and optimizations are defined as plaintext schemas instead of code
Philosophy: Speed

For research and industry alike speed is crucial for state-of-the-art models and massive data
Philosophy: Modularity

New tasks and settings require flexibility and extension.
Philosophy: Openness

Scientific and applied progress call for common code, reference models, and reproducibility.
Philosophy: Community

Academic research, startup prototypes, and industrial applications all share strength by joint discussion and development in a BSD-2 project.
Major Highlights:

Modularity

- Designed to be as modular as possible
  - i.e. allows easy extension to new data formats, network layers and loss functions

- Lot’s of this layers are already implemented.

- Plentiful examples are available to show how to compose these layers
Major Highlights:

Separation of representation and implementation

- Caffe model definitions are written as config files using the Protocol Buffer language
- Switching between a CPU and GPU implementation is exactly one function call
- Caffe supports network architectures in the form of arbitrary directed acyclic graphs. Upon instantiation
Major Highlights:

Test coverage

- Every single module in Caffe is tested
- No new code is accepted without their corresponding tests
- Therefore this helps in refactoring and rapid improvements in the codebase.
Major Highlights:

- Python and MATLAB bindings
- Pre-trained reference models
Software Architecture

- Data Storage
- Layers
- Networks and Run Mode
- Training a Network
Data Storage

- Caffe stores stores and communicates data using blobs.
- Models are stored to disk as Google Protocol Buffers
- Large-scale data is stored in LevelDB databases
Blobs

- Caffe stores, communicates, and manipulates the information as *blobs*

- Blobs provide a unified memory interface holding data; e.g., batches of images, model parameters, and derivatives for optimization.
Blobs

- A Blob is a wrapper over the actual data being processed and passed along by Caffe
- Under the hood provides synchronization capability between the CPU and the GPU
- Mathematically, a blob is an N-dimensional array stored in a C-contiguous fashion.
Blobs

- Conventionally the data is stored in 4D arrays
- Why 4D arrays?
Blobs

- Blobs conceal the computational and mental overhead of mixed CPU/GPU operation by synchronizing from the CPU host to the GPU device as needed.

- In practice, one loads data from the disk to a blob in CPU code, calls a CUDA kernel to do GPU computation, and ferries the blob off to the next layer, ignoring low-level details while maintaining a high level of performance.

- Memory on the host and device is allocated on demand (lazily) for efficient memory usage.
Google Protocol Buffers

- Protocol buffers are a flexible, efficient, automated mechanism for serializing structured data – think XML, but smaller, faster, and simpler.
Google Protocol Buffers

 Protocol buffers are a flexible, efficient, automated mechanism for serializing structured data – think XML, but smaller, faster, and simpler.

- Useful for communication over wire and storing data
- Produces minimal sized binary strings when serialized
- Efficient serialization
- A human-readable text format compatible with the binary version
- Efficient interface implementations in multiple languages, most notably C++ and Python.
LevelDb

- LevelDB is a key/value store built by Google
- Does not provide a server or command line interfaces
- Used as a library stores data in files
- Any guesses who the creators of the LevelDB are?
Layers

- The layer is the essence of a model and the fundamental unit of computation
- Layers do the heavy lifting, like
  - Convolve filters
  - Pool
  - Apply nonlinearities like ReLU etc.
Layers

- A layer takes input through *bottom* connections and makes output through *top* connections.
- It takes one or more blobs as input, and yields one or more blobs as output.
Layers

- Two major responsibility.

- A *forward pass* that takes the inputs and produces the outputs

- A *backward pass* that takes the gradient with respect to the output, and computes the gradients with respect to the parameters and to the inputs, which are in turn back-propagated to earlier layers.
Networks and Run Mode

- The net is a set of layers connected in a computation graph – a directed acyclic graph (DAG) to be exact.

- Caffe does all the bookkeeping for any directed acyclic graph of layers, ensuring correctness of the forward and backward passes.

- A typical net begins with a data layer that loads from disk and ends with a loss layer that computes the objective for a task such as classification or reconstruction.

- The network is run on CPU or GPU by setting a single switch.

- Layers come with corresponding CPU and GPU routines that produce identical results.

- The CPU/GPU switch is seamless and independent of the model definition.
Training a Network using **Solver**

- The solver orchestrates model optimization by coordinating the network’s forward inference and backward gradients to form parameter updates that attempt to improve the loss.

- Some of the Caffe solvers are:
  - Stochastic Gradient Descent
  - AdaDelta
  - Adaptive Gradient etc.
Solver

- Scaffolds the optimization bookkeeping and creates the training network for learning and test network(s) for evaluation.
- Iteratively optimizes by calling forward / backward and updating parameters
- Snapshots the model and solver state throughout the optimization
Solver

- Calls network forward to compute the output and loss
- Calls network backward to compute the gradients
- Incorporates the gradients into parameter updates according to the solver method
- Updates the solver state according to learning rate, history, and method
Quick Recap of Caffe

- Nets, Layers, and Blobs: the anatomy of a Caffe model.
- Forward / Backward: the essential computations of layered compositional models.
- Loss: the task to be learned is defined by the loss.
- Solver: the solver coordinates model optimization.
- Layer Catalogue: the layer is the fundamental unit of modeling and computation – Caffe’s catalogue includes layers for state-of-the-art models.
- Interfaces: command line, Python, and MATLAB Caffe.
- Data: how to caffeinate data for model input.
Tutorial

Logistic Regression
Define the Network

We have to define everything as part of layers. Each layer is JSON type structure.
Define the Network

The key word 'top' is equivalent to output of the layer.

The key word 'bottom' represents the input of the layer.

```yaml
name: "LogReg"
layer {
    name: "mnist"
    type: "Data"
    top: "data"
    top: "label"
    data_param {
        source: "input_leveldb"
        batch_size: 64
    }
}
layer {
    name: "ip"
    type: "InnerProduct"
    bottom: "data"
    top: "ip"
    inner_product_param {
        num_output: 2
    }
}
layer {
    name: "loss"
    type: "SoftmaxWithLoss"
    bottom: "ip"
    bottom: "label"
    top: "loss"
}
```

Diagram:

- **mnist (Data)**
- **data**
- **ip (INNER_PRODUCT)**
- **ip**
- **loss (SOFTMAX_LOSS)**
- **label**
Define the network

Save the network in a prototxt file (which is the Google’s ProtoBuf File)

Once the network is defined, now define the solver
Example Solver

```
net: "models/train_val.prototxt" # path to the network definition
test_iter: 200 # how many mini-batches to test in each validation phase
test_interval: 500 # how often do we call the test phase
base_lr: 1e-5 # base learning rate
lr_policy: "step" # step means to decrease lr after a number of iterations
gamma: 0.1 # ratio of decrement in each step
stepsize: 5000 # how often do we step (should be called step_interval)
display: 20 # how often do we print training loss
max_iter: 450000
momentum: 0.9
weight_decay: 0.0005 # regularization!
snapshot: 2000 # taking snapshot is like saving your progress in a game
snapshot_prefix: "models/model13_train_0422"
solver_mode: GPU
```
Provide the path where the definition of the model is defined.

```plaintext
net: "models/train_val.prototxt" # path to the network definition

test_iter: 200 # how many mini-batches to test in each validation phase
test_interval: 500 # how often do we call the test phase
base_lr: 1e-5 # base learning rate
lr_policy: "step" # step means to decrease lr after a number of iterations
gamma: 0.1 # ratio of decrement in each step
stepsize: 5000 # how often do we step (should be called step_interval)
display: 20 # how often do we print training loss
max_iter: 450000
momentum: 0.9
weight_decay: 0.0005 # regularization
snapshot: 2000 # taking snapshot is like saving your progress in a game
snapshot_prefix: "models/model3_train_0422"
solver_mode: GPU
```
Solver

Provide all the parameters which are required for the learning and optimization.

```
net: "models/train_val.prototxt" # path to the network definition

test_iter: 200 # how many mini-batches to test in each validation phase

net: "models/train_val.prototxt" # path to the network definition

test_interval: 500 # how often do we call the test phase

net: "models/train_val.prototxt" # path to the network definition

base_lr: 1e-5 # base learning rate

net: "models/train_val.prototxt" # path to the network definition

lr_policy: "step" # step means to decrease lr after a number of iterations

net: "models/train_val.prototxt" # path to the network definition

gamma: 0.1 # ratio of decrement in each step

net: "models/train_val.prototxt" # path to the network definition

steps: 5000 # how often do we step (should be called step_interval)

net: "models/train_val.prototxt" # path to the network definition

display: 20 # how often do we print training loss

net: "models/train_val.prototxt" # path to the network definition

max_iter: 450000

net: "models/train_val.prototxt" # path to the network definition

momentum: 0.9

net: "models/train_val.prototxt" # path to the network definition

weight_decay: 0.0005 # regularization!

net: "models/train_val.prototxt" # path to the network definition

snapshot: 2000 # taking snapshot is like saving your progress in a game

net: "models/train_val.prototxt" # path to the network definition

snapshot_prefix: "models/model3_train_0422"

net: "models/train_val.prototxt" # path to the network definition

solver_mode: GPU
```
Solver

Provide the mode to run, GPU or CPU

```plaintext
net: "models/train_val.prototxt"  # path to the network definition

solver_mode: GPU
```
Solver

Save the solver in a protobuf file, for example: solver.prototxt
Finally How to Run?

!./build/tools/caffe train -solver solver.prototxt
Include the test and train layers separately.
More Examples

We can define the convolution layer’s parameters as kernel seize, stride etc.
More Examples

AlexNet implementation
Caffe Pros

- Good for feedforward networks
- Train models without writing any code!
- Excellent ConvNet implementation
- Good for feedforward and networks and image processing
Caffe Cons

- Need to write C++ / CUDA for new GPU layers
- Not good for recurrent networks
- Cumbersome for big networks (GoogLeNet, ResNet)
  - But we can generate the definitions from Python interface
Caffe 2 vs Caffe 1

- First-class support for large-scale distributed training
- Mobile deployment
- New hardware support (in addition to CPU and CUDA)
- Flexibility for future directions such as quantized computation
- Stress tested by the vast scale of Facebook applications
Operators in Caffe 2

- One of the basic units of computation in Caffe2 are the Operators

- Each operator contains the logic necessary to compute the output given the appropriate number and types of inputs and parameters

- The overall difference between operators’ functionality in Caffe and Caffe2 is illustrated in the following graphic, respectively
Versions of Caffe

- NVIDIA Caffe
  - Tunes for NVIDIA GPU’s particularly with multi GPU configurations

- Intel Caffe
  - Dedicated to improving Caffe performance when running on CPU, in particular Intel® Xeon processors

- OSU Caffe
  - Brings DL-Awareness to the MPI runtime by designing efficient CUDA-Aware collective operations for very large messages

- and many more from Yahoo on running for Spark etc.
Caffe at Facebook

- In production for vision at scale, uploaded photos run through Caffe
- Automatic Alt Text for the blind
- On This Day for surfacing memories
- Objectionable content detection
- Contributing back to the community: inference tuning, tools, code review include fb-cafe-exts
Caffe at Pinterest

- In production for vision at scale uploaded photos run through Caffe
- Deep learning for visual search: retrieval over billions of images in <250 ms
- ~4 million requests/day
- Built on an open platform of Caffe, FLANN, Thrift, ...
Caffe at Yahoo

- Curate news and restaurant photos for recommendation
- Arrange user photo albums
- Integrated Caffe with Apache Spark to create CaffeOnSpark, a distributed deep learning framework
News on Caffe

Caffe 2 was merged into PyTorch in March 2018 by Facebook
Thank You!