CS 6423
Scalable Computing for Data Analytics

Lecture 11: Overview

Prof. Gregory Provan
Department of Computer Science
University College Cork

Overview

- Computational platforms for deep learning
- Underlying computational framework
  - Computation graph
- Comparison of tools
- In-depth: Caffe2

TensorFlow vs. Caffe

- **Layer-wise design**
  A neural network is a computational graph. In caffe, each node is a layer. In TensorFlow, each node is a tensor operation (e.g. matrix add/multiply, convolution, etc.). A layer can be defined as a composition of those operations. What this means is that the building brick in TensorFlow is smaller than the building brick in caffe. That's why caffe is not considered flexible because for new layer types (which can be composed using existing bricks in TensorFlow), you have to define the full forward, backward, and gradient update. You can see an already long-list of layers implemented in (official) caffe.
- What's worse is that if you want to support both CPU and GPU, you need to implement extra functions, e.g. *Forward_gpu and Backward_gpu*.
- Worse, you need to assign an *int* id to your layer type and add that to the proto file. If your pull request is not merged early, you may need to change the id because someone else already claims that. [*]

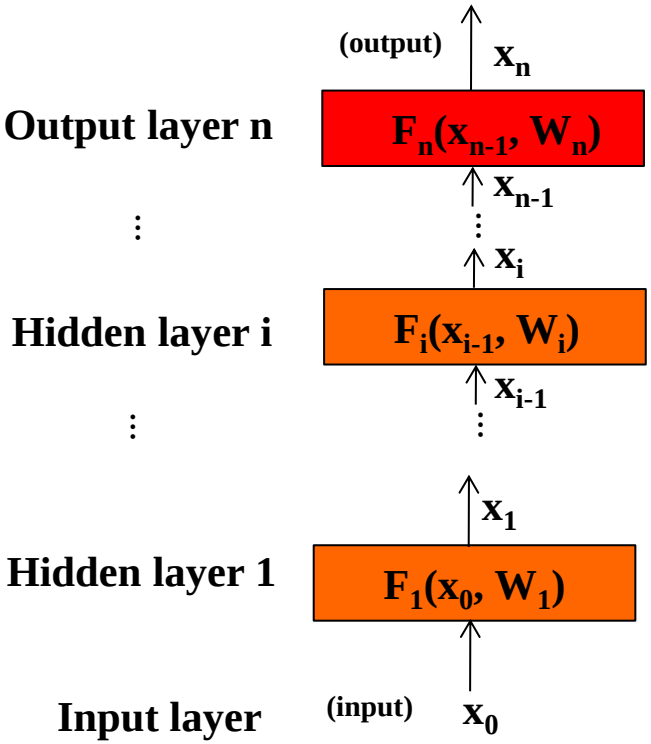
Framework Comparison

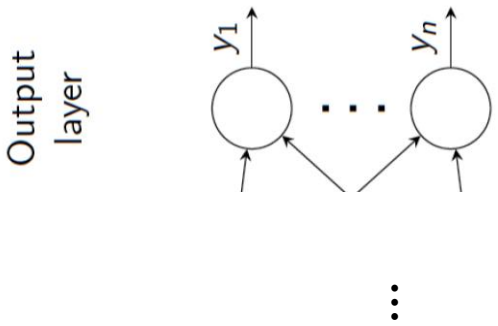
- More alike than different
  - All express deep models
  - All are open-source (contributions differ)
  - Most include scripting for hacking and prototyping
  - Many have same computational basis: computation graph

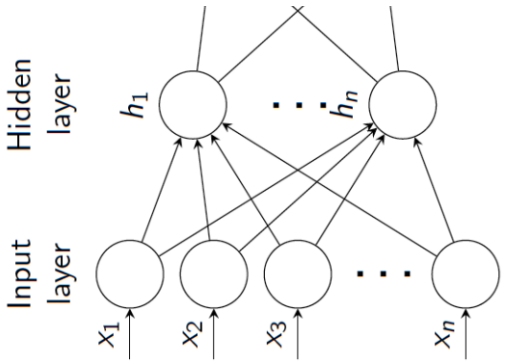
- No strict winners – experiment and choose the framework that best fits your work

Computation Graph: Basis

- Decompose inference into elements
- Tools differ by granularity of elements

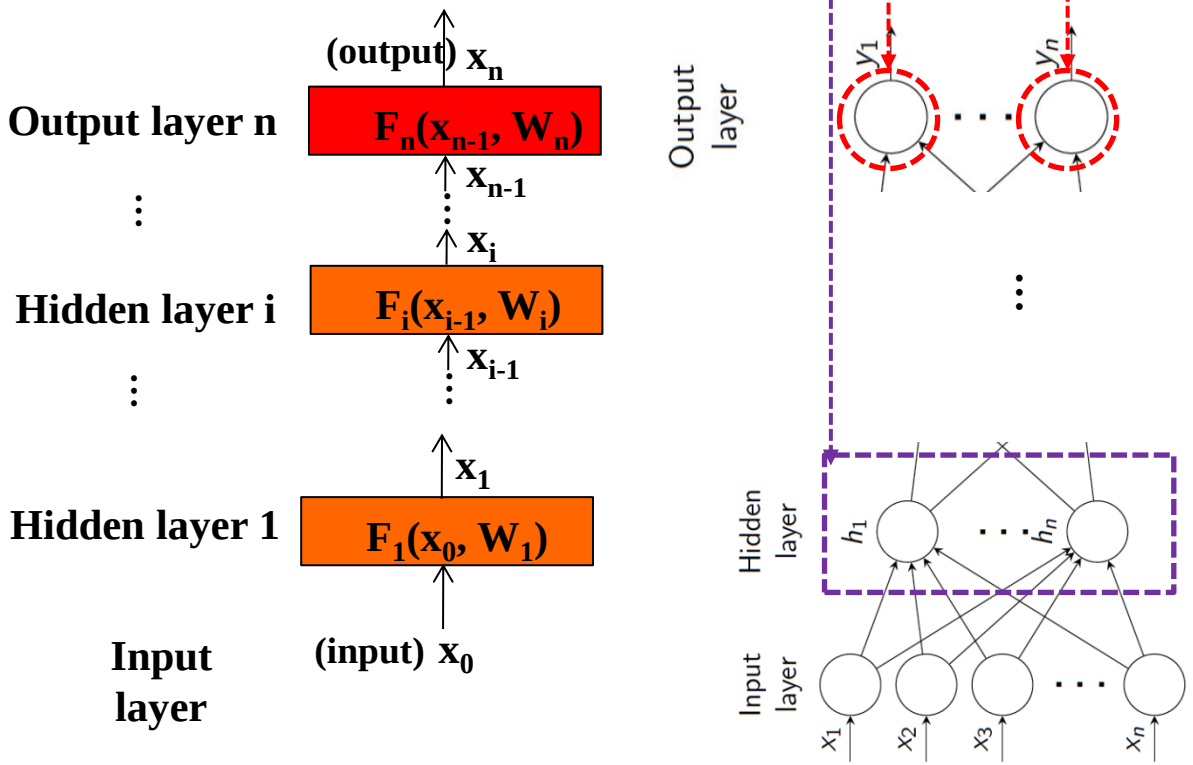






Examine Two Approaches

- TensorFlow
  - Fine granularity: computation graph
- Caffe2
  - Granularity: “level” of inference



So what is Caffe?

- Pure C++ / CUDA architecture for deep learning
  - command line, Python, MATLAB interfaces
- Fast, well-tested code
- Tools, reference models, demos, and recipes
- Seamless switch between CPU and GPU
  - `Caffe::set_mode(Caffe::GPU);`

Prototype

Training

Deployment

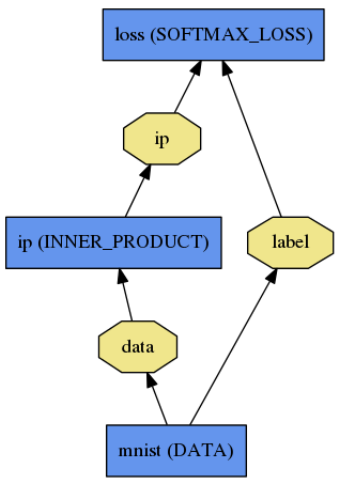
All with essentially the same code!

Network: Computation Graph

- A network is a set of layers connected as a DAG:

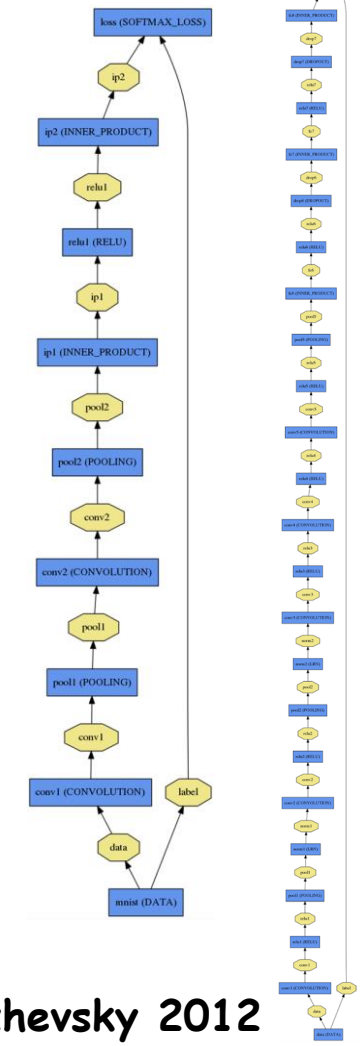
```
name: "dummy-net"
layers { name: "data" …}
layers { name: "conv" …}
layers { name: "pool" …}
    … more layers …
layers { name: "loss" …}
```

- Caffe creates and checks the net from the definition.
- Data and derivatives flow through the net as *blobs* – a an array interface



LogReg

↑



LeNet

→

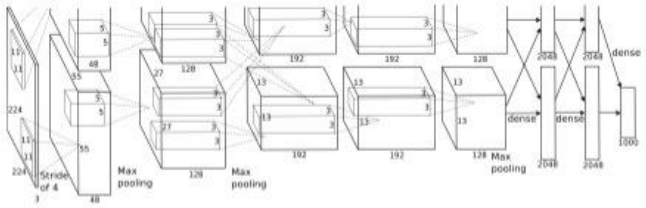
ImageNet, Krizhevsky 2012

→

Forward / Backward

the essential Net computations

Forward: inference $f_W(x)$



"espresso" + loss

$\nabla f_W(x)$ Backward: learning

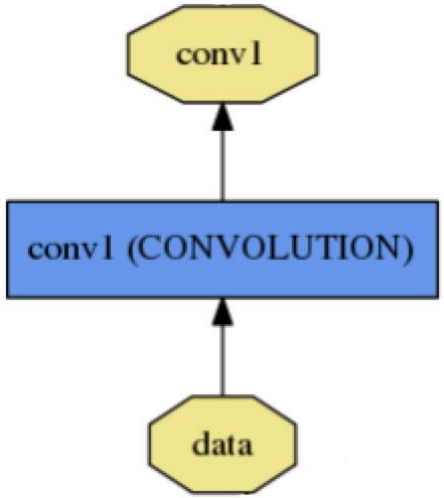
Caffe models are complete machine learning systems for inference and learning. The computation follows from the model definition. Define the model and run.

Representing Layers

```
name: "conv1"
type: CONVOLUTION
bottom: "data"
top: "conv1"
convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
        type: "xavier"
    }
}
```

name, type, and the connection structure (input blobs and output blobs)

layer-specific parameters



- Every layer type defines
  - Setup
  - Forward
  - Backward

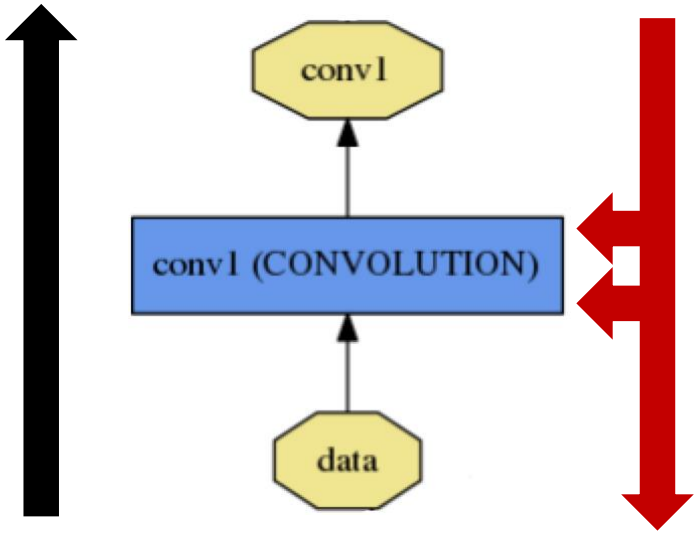
* Nets + Layers are defined by protobuf schema

Layer Protocol

Setup: run once for initialization.

Forward: make output given input.

Backward: make gradient of output
- w.r.t. bottom
- w.r.t. parameters (if needed)



Model Composition
The Net forward and backward passes are the composition the layers'.

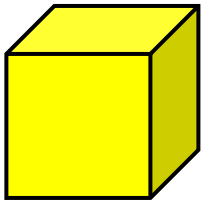
Layer Development Checklist

Tensor Representation: Blob

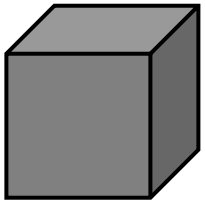
```
name: "conv1"
type: CONVOLUTION
bottom: "data"
top: "conv1"
… definition …
```

Blobs are 4-D arrays for storing and communicating information.

- **hold data, derivatives, and parameters**
- **lazily allocate memory**
- **shuttle between CPU and GPU**



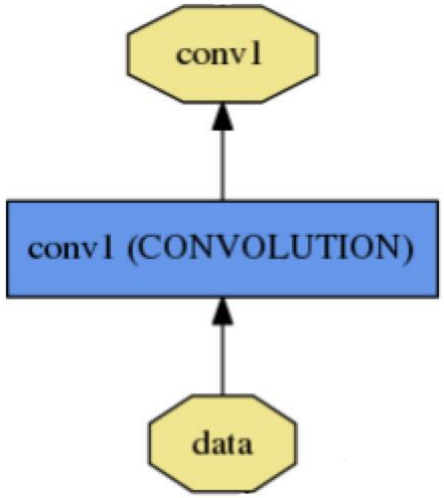
Data
Number x *K* Channel x *Height* x *Width*
256 x 3 x 227 x 227 for ImageNet train input



Parameter: Convolution Weight
N Output x *K* Input x *Height* x *Width*
96 x 3 x 11 x 11 for CaffeNet conv1

Parameter: Convolution BIas
96 x 1 x 1 x 1 for CaffeNet conv1

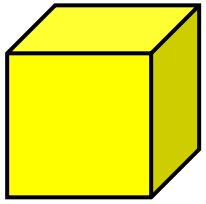
top blob



bottom blob

Tensor Representation: Blob

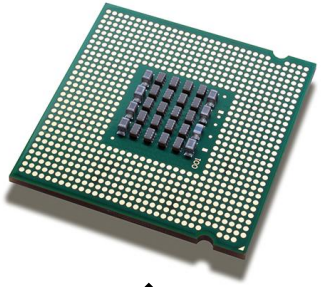
Blobs provide a unified memory interface.



Reshape(num, channel, height, width)
- declare dimensions
- make *SyncedMem* -- but only lazily allocate

cpu_data(), mutable_cpu_data()
- host memory for CPU mode
gpu_data(), mutable_gpu_data()
- device memory for GPU mode

{cpu,gpu}_diff(), mutable_{cpu,gpu}_diff()
- derivative counterparts to data methods
- easy access to data + diff in forward / backward

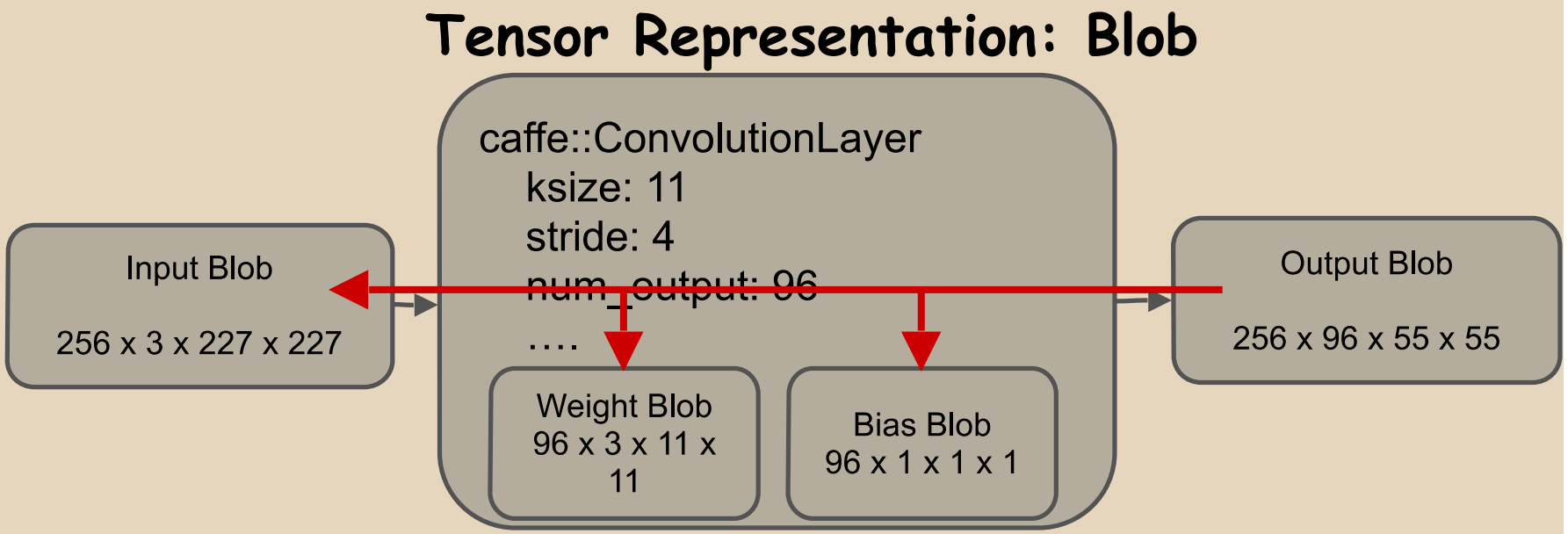


SyncedMem allocation + communication

High-Level Representation

High-Level Representation

- Forward: given input, computes the output. →
- Backward: given the gradient w.r.t. the output, compute the gradient w.r.t. the input and its internal parameters. →
- Setup: how to initialize the layer.



GPU/CPU Switch with Blob

- Use synchronized memory
- Mutable/non-mutable determines whether to copy

Types of Layers

- Data layers
- Vision layers
- Common layers
- Activation/Neuron layers
- Loss layers

Data Layers

- Data enters through data layers -- they lie at the bottom of nets.
- Data can come from efficient databases (*LevelDB* or LMDB), directly from memory, or, when efficiency is not critical, from files on disk in HDF5/.mat or common image formats.
- Common input preprocessing (mean subtraction, scaling, random cropping, and mirroring) is available by specifying `TransformationParameters`.

More about Layers

- Data layers
- Vision layers
- Common layers
- Activation/Neuron layers
- Loss layers

Vision Layers

- Images as input and produce other images as output.
- Non-trivial height h>1 and width w>1.
- 2D geometry naturally lends itself to certain decisions about how to process the input.

- In particular, most of the vision layers work by applying a particular operation to some region of the input to produce a corresponding region of the output.
- In contrast, other layers (with few exceptions) ignore the spatial structure of the input, effectively treating it as "one big vector" with dimension "***chw***".

Vision Layers

- Convolution
- Pooling
- Local Response Normalization (LRN)
- Im2col -- helper

More about Layers

- Data layers
- Vision layers
- Common layers
- Activation/Neuron layers
- Loss layers

Common Layers

- `INNER_PRODUCT` $W^T x+b$ (fully conn)
- `SPLIT`
- `FLATTEN`
- `CONCAT`
- `SLICE`
- `ELTWISE` (element wise operations)
- `ARGMAX`
- `SOFTMAX`
- `MVN` (mean-variance normalization)

More about Layers

- Data layers
- Vision layers
- Common layers
- Activation/Neuron layers
- Loss layers

Activation/Neuron layers

- One Input Blob
- One Output Blob
  - Both same size

Activation/Neuron layers

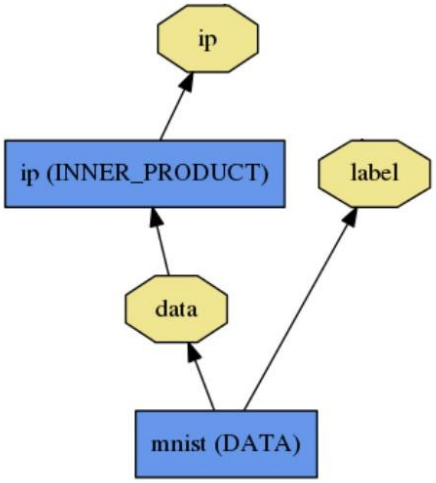
- RELU
- SIGMOID
- TANH
- ABSVAL
- POWER
- BNLL (binomial normal log likelihood)

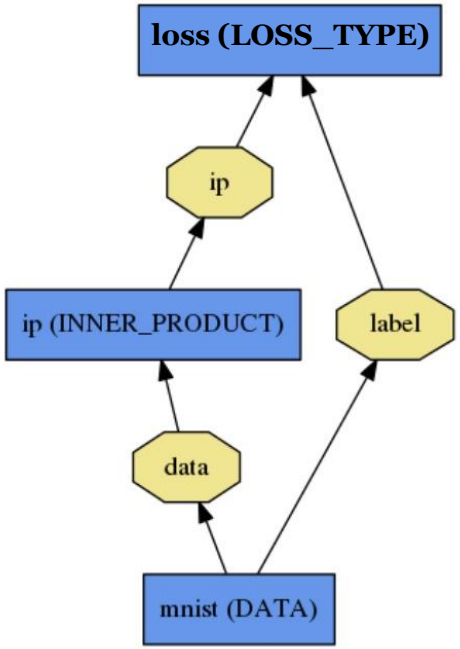
More about Layers

- Data layers
- Vision layers
- Common layers
- Activation/Neuron layers
- Loss layers

Loss

What kind of model is this?





Who knows! Need a loss function.

Classification
`SOFTMAX_LOSS`
`HINGE_LOSS`

Linear Regression
`EUCLIDEAN_LOSS`

Attributes / Multiclassificatio
`SIGMOID_CROSS_ENTROPY_LOSS`

Others…

New Task
`NEW_LOSS`

Loss

- **Loss function** determines the learning task.
- Given data D, a Net typically minimizes:

$$L(W) = \frac{1}{|D|} \sum_{i}^{|D|} f_W\left(X^{(i)}\right) + \lambda r(W)$$

Data term: error averaged over instances

Regularization term: penalize large weights to improve generalization

Loss

- The data error term $f_W\left(X^{(i)}\right)$ is computed by Net::Forward
- Loss is computed as the output of Layers
- Pick the loss to suit the task – many different losses for different needs