ML4 - Caffe2

Group 04 - 21KHMT02

21°

Table of contents

01

Introduction

Introduction of our group members

»»» Group members

- 21127471 - Nguyễn Hoàng Anh Tuấn
- 21127687 - Phan Huy Đức Tài
- 21127605 - Dương Gia Hân
- 21127676 - Phan Hữu Phước

Meet us

02

Problem statement

What prompted the invention of Caffe2?

A brief history of deep learning

1940s: Origin of the concept "deep learning"

1970s: Birth of CNNs and practical applications of deep learning

1990s: New neural network architectures

2010s - ?: Rapid growth of deep learning in research and applications

What seems to be the problem?

- Replication of neural networks is time-consuming
- Non-standardized GPU configuration for training
- Hindered research & development progress

03

What is Caffe2?

An in-depth description of Caffe2 and its applications

»»» Development history

- In 2014, Dr. Yangqing Jia proposed (and implemented) a framework aimed to help build neural networks in his PhD thesis.
- The Caffe paper was released alongside his thesis, which stole the attention of many researchers and developers.
- Caffe was later maintained by BAIR and a few volunteers at the lab.

Dr. Yangqing Jia

»»» Development history

- In 2017, Facebook announced Caffe2, a superior version of its predecessor both in efficiency and structural design.
- Upon PyTorch's newer releases, Caffe2 was announced its "death", and since then merged in PyTorch's source code.
- Caffe2's API still works, somewhat. Hence a better replacement would be to convert to PyTorch fully.

Why Caffe2?

Open-source

- Written in C++, optimized CUDA for training.
- Supports Python, MATLAB
- Supports distributed training

Mobile deployement

- Excellent support for mobile deployment on iOS and Android
- Stress-tested by Facebook

Large community

- Caffe Model Zoo contains pretrained models from developers.
- Official website also provides useful training datasets.

Caffe2 components

Workspaces

How Caffe2 manages memory and excution of net

Operators

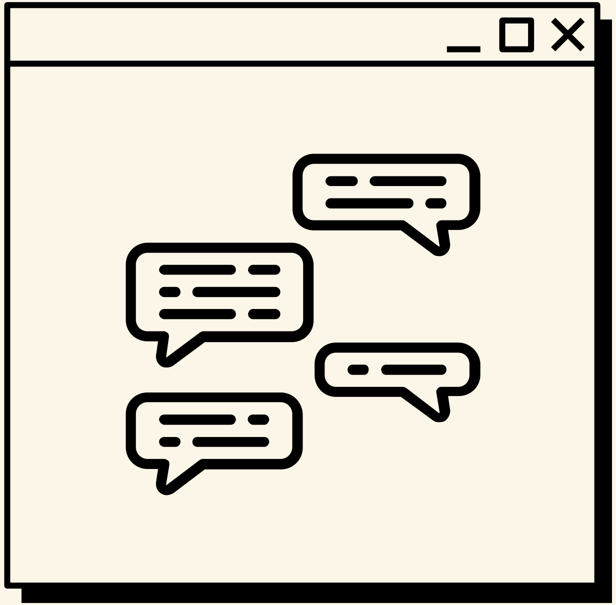
The core of Caffe2 framework

Nets

Complete computational graphs

Workspaces

- Caffe2 lets user freely interacts with multiple workspaces.
- The workspace consists of blobs, which is usually a N-dimensional tensor to numpy’s ndarray, however contiguous.
- Blobs are actually typed pointer that can store any C++ objects, proving its usefulness in memory related manipulations.
- It handle the execution of Model schema defined by programmer.



Examples of workspaces

- All workspaces when initialized are empty of Blobs. We can feed blobs into a workspace by using the command FeedBlob().

```
X = np.random.randn(2, 3).astype(np.float32)
print("Generated X from numpy:\n{}".format(X))
workspace.FeedBlob("X", X)
```

```
Generated X from numpy:
[[ 1.0922441  -0.65129787 -0.2511869 ]
 [ 1.3999398  -0.86516035 -2.0602188 ]]
Out[3]:
True
```

- By running the code that list all blobs in the workspace, we get the following results:

```
print("Current blobs in the workspace: {}".format(workspace.Blobs()))
print("Workspace has blob 'X'? {}".format(workspace.HasBlob("X")))
print("Fetched X:\n{}".format(workspace.FetchBlob("X")))
```

```
Current blobs in the workspace: [u'X']
Workspace has blob 'X'? True
Fetched X:
[[ 1.0922441  -0.65129787 -0.2511869 ]
 [ 1.3999398  -0.86516035 -2.0602188 ]]
```

Examples of workspaces

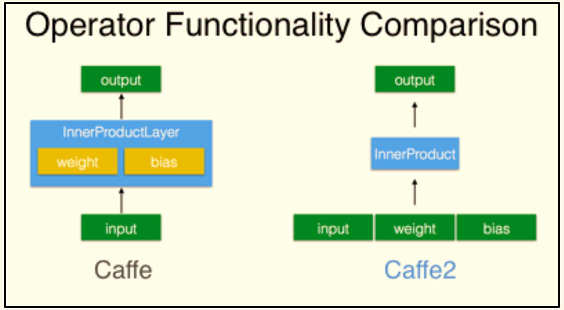
- If we were to switch to another workspace, the **X** blob created in the previous example would not exist.

```
print("Current workspace: {}".format(workspace.CurrentWorkspace()))
print("Current blobs in the workspace: {}".format(workspace.Blobs()))

# Switch the workspace. The second argument "True" means creating
# the workspace if it is missing.
workspace.SwitchWorkspace("gutentag", True)

# Let's print the current workspace. Note that there is nothing in the
# workspace yet.
print("Current workspace: {}".format(workspace.CurrentWorkspace()))
print("Current blobs in the workspace: {}".format(workspace.Blobs()))
```

```
Current workspace: default
Current blobs in the workspace: [u'X']
Current workspace: gutentag
Current blobs in the workspace: []
```

Operators

- Operators in Caffe2 are functions in its core. Written in C++ aside, they all derive from a common interface, registered by type, so we can call different operators during runtime.
- When an Operator is created, nothing is run. The program simply creates the protocol buffer that specifies what operator should be executed.
- These protocol buffers are sent to C++ backend for execution later



Examples of Operators

- We can create an Operator and verify its existence in the workspace with the following snippet

```python
op = core.CreateOperator(
    "Relu", # The type of operator that we want to run
    ["X"], # A list of input blobs by their names
    ["Y"], # A list of output blobs by their names
)
print("Type of the created op is: {}".format(type(op)))
print("Content:\n")
print(str(op))
```

- Which should yield the following results

```
Type of the created op is: <class 'caffe2.proto.
Content:

input: "X"
output: "Y"
name: ""
type: "Relu"
```

Examples of Operators

- As mentioned, it will not execute the ReLU operator, as no input was passed. To activate the operator, we can do the following

```
workspace.FeedBlob("X", np.random.randn(2, 3).astype(np.float32))
workspace.RunOperatorOnce(op)
```

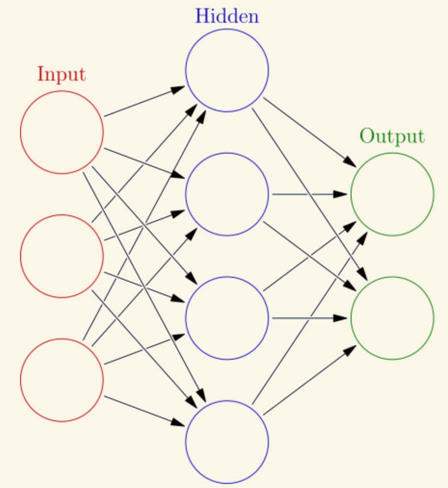
- If we run the code to check the blobs in our workspace and their values, we should be getting the following results

```
print("Current blobs in the workspace: {}\n".format(workspace.Blobs()))
print("X:\n{}\n".format(workspace.FetchBlob("X")))
print("Y:\n{}\n".format(workspace.FetchBlob("Y")))
print("Expected:\n{}\n".format(np.maximum(workspace.FetchBlob("X"), 0)))
```

```
Current blobs in the workspace: [u'X', u'Y']

X:
[[-0.958609   -1.5830146  -1.1998546 ]
 [ 0.77405983  0.08448339  0.57201284]]

Y:
[[0.         0.         0.        ]
 [0.77405983 0.08448339 0.57201284]]

Expected:
[[0.         0.         0.        ]
 [0.77405983 0.08448339 0.57201284]]
```

Nets

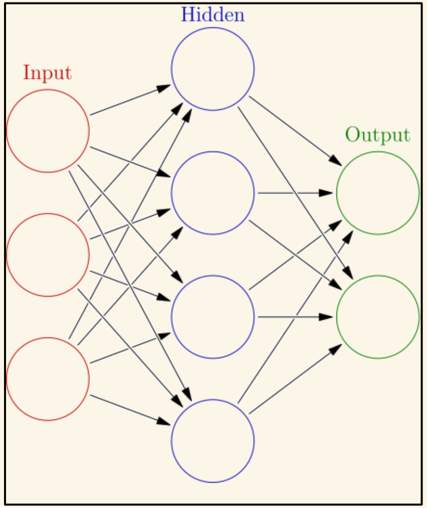
- Essentially, nets are computational graphs. The name is supposed to represent the familiar term neural networks.
- A Net is composed of multiple Operators written sequentially, which forms a series of commands.



Examples of Nets

- From the examples above, we can already form a workspace and put data in blobs. Here we will create a simple neural network architecture equivalent to the following code

```
X = np.random.randn(2, 3)
W = np.random.randn(5, 3)
b = np.ones(5)
Y = X * W^T + b
```



Examples of Nets

- We can start by defining our **core.Net** object, which is a wrapper class around a NetDef protocol buffer

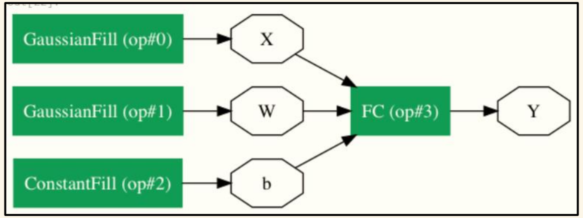
```
net = core.Net("my_first_net")
print("Current network proto:\n\n{}".format(net.Proto()))
```

- Now we can define our W, b, and X values

```
X = net.GaussianFill([], ["X"], mean=0.0, std=1.0, shape=[2, 3], run_once=0)
W = net.GaussianFill([], ["W"], mean=0.0, std=1.0, shape=[5, 3], run_once=0)
b = net.ConstantFill([], ["b"], shape=[5,], value=1.0, run_once=0)
```

- And set Y according to the formula

```
Y = X.FC([W, b], ["Y"])
```



Comparison with other frameworks

Feature	Caffe	TensorFlow	Caffe2	MXNet	Shogun
Speed and Memory Usage During Training	Renowned for its speed, particularly in image classification and segmentation tasks.	Generally considered slower in comparison to Caffe when it comes to training but offers a more flexible environment.	Known for its focus on speed and efficiency, especially for mobile and embedded devices.	Offers good balance between speed and memory usage.	Primarily focused on machine learning algorithms rather than deep learning. However, it can be memory-efficient due to its C++ core.
Popularity (Community & Documentation)	Popular in startups, academic research projects, and industrial applications in computer vision, speech, and multimedia. Small yet active community	Widely adopted and has a massive community. Extensive resources, tutorials, and pre-trained models available. Used by researchers, developers, and enterprises worldwide.	Less popular than TensorFlow but gaining traction. Community size is smaller compared to TensorFlow.	More widely used with a larger and active community. This translates to better documentation, tutorials, and easier finding help online.	Primarily used in the academic research community. Documentation might be geared towards researchers.

Comparison with other frameworks

Feature	Caffe	TensorFlow	Caffe2	MXNet	Shogun
Learning Curve	Considered user-friendly and easy to use, making it accessible for beginners, especially with its focus on vision tasks.	Has a steeper learning curve due to its comprehensive and sometimes complex functionalities.	As part of PyTorch, offers a balance between ease of use and advanced features, which can be appealing for both beginners and experienced users.	MXNet has a higher-level abstraction like Gluon for deep learning tasks, making it easier to learn.	Steeper learning curve due to its C++ core and focus on algorithms.
Price	Open-source and free to use. No direct costs associated with licensing.	Open-source and free. However, consider infrastructure costs (e.g., GPUs, cloud resources) for large-scale deployments	Also open-source and free. Similar considerations apply regarding infrastructure costs.	Open-source and free to use.	Open-source and free.

04

Practical demo

How Caffe2 is applied in the modern day.

References

- https://arxiv.org/pdf/1406.2199.pdf,
- Module: tf | TensorFlow v2.15.0.post1
- Keras 3 API documentation
- Operators Overview | Caffe2
- https://github.com/woodfrog/ActionRecognition
- A Brief History of Deep Learning - DATAVERSITY

Thanks for listening!

Does anyone have any questions?