Comparison of deep learning frameworks

Marin Jukic
Fachhochschule Aachen
Aachen, Germany
marin.jukic@alumni.fh-aachen.de

Abstract

Since the capabilities of deep learning applications have grown more and more, there appeared a variety of different deep learning framework. They have all their strength and weaknesses for the different areas of applications with their different approaches. In this paper, we take a look on the current deep learning frameworks and compare the most popular with each other. The comparison is carried out based on various aspects such as distributed training and visualization. With this comparison, the differences between the frameworks can be shown. It will also address other aspects such as GPU-support and deployment. The results of the comparison provide an opportunity to make an informed choice of a deep learning framework for a use case.

Keywords
Deep learning frameworks, Distributed training, ONNX, Deployment

I. INTRODUCTION

In recent years, deep learning methods have proven to be very useful with great success. They have achieved good results and big steps in several domains like computer vision, natural langue processing and speech recognition. [1] Thus, various deep learning frameworks have been developed by the industry and community during that time in order to use the deep learning methods effectively in software environments. It is also easier to have a first contact with those methods using a practical software framework than to experience them theoretically.

The decision of the right deep learning framework for the specific use case is essential. If the unsuitable framework is selected, this can affect the required computing and development time and the quality of the results. In addition, the selected framework must be compatible with the given development environment, for example training on a multi node system requires the framework to support distributed training. The right selection can simplify the development and training of the deep learning models and make them more efficient.

There are many open-source deep learning frameworks in the industry. Some of them were developed by big companies (like TensorFlow by Google) and other were developed during an open-source project. First, we look in section II on the different deep learning frameworks, which are well known in the industry. Then we define the criteria for the comparison in section III. In section IV the comparison of the frameworks (which are currently maintained and developed) is carried out. Finally, a conclusion of the comparison is drawn in section V.

II. DEEP LEARNING FRAMEWORKS

There are a many deep learning framework and libraries in the industry. There have all their advantages and disadvantages. First, we look on some of the most known frameworks.

Frameworks:

- TensorFlow is an end-to-end open-source machine and deep learning framework. It has a flexible ecosystem of tools, easy model building, and a robust machine learning production on diverse environments (like in the cloud, in the browser or on-device). TensorFlow was developed and maintained by the Google Brain team within Google. It is currently published under the version 2.3.0. as an open-source project for machine learning and deep learning domain. [3]
- PyTorch is an open-source framework for GPU-accelerated deep learning that is based on the framework Torch. It supports distributed training, has a robust ecosystem, is production ready, and supported by the major cloud platforms. It was created by Facebook's AI research group in 2016 and is since then maintained by them. PyTorch has gained a lot in popularity in the recent years since its debut. [4]
- Apache MXNet is a deep learning framework aimed for efficiency and flexibility. It also features high-level APIs. It is supported by major public cloud providers (like AWS). [2] [5]
- Microsoft Cognitive Toolkit (CNTK) is an open-source framework for commercial-grade deep learning developed by Microsoft. It can be included as a library or used as a standalone machine-learning tool through its own model description language BrainScript. [6]
- Caffe is a deep learning framework developed by Berkeley AI Research and by community contributors. It has expression, speed, and modularity in its focus. [7] Currently it is not actively developed. Its successor Caffe2 is being merged with PyTorch. [12]
- Chainer is a powerful, flexible, and intuitive deep learning framework developed by Preferred Networks [8]. It is based on the "Define-and-Run" scheme rather than separate the definition and the computation of the graph. With this scheme the network is defined dynamically via the actual forward computation.
- DeepLearning4j (DL4J) is different from the other deep learning/machine learning frameworks in specific with its API languages. It is an open source, distributed deep learning framework implemented for Java. Furthermore, it is aimed to the industrial Java development ecosystem and Big Data processing (for example with Spark and Hadoop). [9]
- Theano is a deep learning tool/library with GPU and multi-GPU support. It is an open-source project under

the BSD license. Theano allows to define, optimize, and evaluate efficiently mathematical expression into multi-dimensional array. Currently it is maintained but not actively developed. [2]

DL Framework	Github projects
TensorFlow	98012
PyTorch	56677
Chainer	28111
Caffe	8547
MXNet	2572
Theano	2259
CNTK	554
DeepLearning4j	482

Table 1: Number of GitHub projects. [20]

For the further comparison, we take the 5 most popular and actively developed/maintained frameworks from table 1 which are TensorFlow, MXNet, CNTK, Chainer and PyTorch.

III. Criteria

Before the comparison of the different framework with each other, we need to define the aspect and indicators, with which they can be compared. Previous works have already compared the performance of different frameworks for different use cases with each other. [17][18] Therefore, we will look on the functionalities and differences of the frameworks above with these criteria's:

Documentation: To get to know all aspects of a framework, it would be advantageous to have a good documentation so that you can find help there for specific questions. In addition, tutorials are very important for newcomers and instructions on how to install the framework with its dependencies for their application and operating system.

API-availability: The frameworks offer their functionalities and capabilities for the developers through APIs. Those can go from the language support all the way to the support of different operating environment (like mobile devices). Therefore, a bigger number of APIs is beneficial to cover more use cases.

ONNX-Support: Open Neural Network Exchange is an open format for the representation of deep learning models. It defines a common set of operators and file format. This enables the developer to work with diverse frameworks, tools, runtimes, and compilers.

Visualization and Debugging: Deep learning models and algorithms are complex. To make them and the results more understandable and better interpretable, the process can be represented visually. This can also be used to check the correctness of the construct. The framework should have to have a visualization functionality for this.

Good debugging capabilities are essential during the development phase. With these, the search of sources of error is made easier and it saves the developer time during the development.

Languages supported: In order to be able to address the largest possible developer group, one must also be able to support their operating language. In addition, the popular languages in deep learning such as Python or Java must be supported.

Distributed training: Deep learning tasks are often carried out on a large cluster to boost the performance and speed up the training time. Those large cluster consists of many GPUs and nodes. For this use case, it is important that the framework can be scaled across multiple GPUs and nodes. [1][2]

GPU-Support: There are three manufacturers of dedicated GPUs in the industry: Intel, NVIDIA, and AMD. [10] The major ones are NVIDIA and AMD. NVIDIA provides support for deep learning training using CUDA. AMD has created ROCm, an open-source platform, where the developers can train their models on AMD GPUs and through CUDA on NVIDIAs GPUs. [15] It is currently not possible to train models on Intel's dedicated and non-dedicated GPU.

Deployment: Not only the development of the models is important, but also the deployment of the final model into an existing system or software environment. Thus, the frameworks should support the deployment of the models for various use cases and conditions like the deployment to mobile devices or direct to the cloud.

These criteria were selected because they cover various aspects of usability, manageability and use cases.

IV. Comparison of the Frameworks

During this section we will compare the frameworks in the individual aspects of the predefined criteria.

A. Documentation

All of our frameworks offer documentation. They are detailed, filled with tutorials and instructions for installation for the different operating system. There is also a possibility to give feedback to the developer and maintainer and to participate in community forums. It is also an advantage that the frameworks were developed and maintained by large IT companies (like Google for TensorFlow), as they already have experience with documentation. [3][4][5][6][11] TensorFlow and PyTorch have the best documentation in terms of structure and readability.

B. API-Availibility

- TensorFlow: TensorFlow offers APIs for the support of several languages, TensorFlow Lite for the development on mobile device environment (like Android and iOS), TFX for the support of the deployment of the application model. There are also several APIs for specified use cases like graph computation, debugging, optimization, distributed training, and wrappers for primitive neural net operations. [3]
- MXNet: MXNet offers different APIs. The most interesting are the APIs for distributed training, different optimizers, metrics, and contribution. Here we do not have so many APIs like in TensorFlow and the documentation structure is not as good as by TensorFlow. [5]
- CNTK: For CNTK, the most interesting APIs are the API for debugging, contribution, learning (optimization) and logging. [6]
- Chainer: Chainer offers besides the support of Python and distributed training a bunch of other APIs. The most interesting are the APIs for the optimization,

visualization of computational graph, the training tools, and the support of models from the Caffe framework. [11]

- PyTorch: PyTorch provides APIs beside the supported languages a variety of other functionalities. The most interesting ones are the APIs for bottleneck, optimization, and the hub, which is a pre-trained model repository. [4]

We can see that all frameworks offer the basic APIs for the supported languages, distributed training, and optimization. Then the frameworks differ. TensorFlow, PyTorch an Chainer offer more APIs than CNTK and MXNet. TensorFlow stands out with the APIs for the different environments (like mobile devices), Chainer with the API for the native support for Caffe model and PyTorch with the hub.

C. ONNX-Support

ONNX-support is important, if you want to build a model in one framework and then used this model to continue with a different framework. Thus, the advantages of various frameworks can be used. CNTK, MXNet, Chainer and PyTorch already support ONNX natively. TensorFlow does not support ONNX natively but there are several tools like onnx-tf to make it possible. Therefore, for example, a model can be trained in TensorFlow and this can be transferred to MXNet using the ONNX format. [19]

D. Visualization and Debugging

- TensorFlow: TensorBoard is a visualization toolkit, which can track and visualize metrics like accuracy and loss. It can visualize the model graph, show histograms of biases, weights and other. It also provides projecting embedding to a lower dimensional space. [2]

  TensorFlow provides two possibilities for debugging your model. Either you can debug it with the *TensorBoard Debugger V2*, which is included in the TensorBoard or through the *tf.debugging* module, which provides a lot of functions for troubleshooting. [2]

- MXNet: MXNet does not bring its own board with it and does not have as much possibilities as TensorFlow. However, it offers possibilities of visually representing the generated neural network as a computation graph. This requires Jupyter Notebook and the Graphviz library. [5]

  MXNet has no integrated debugger. Here you can work with the debugger gdb or with language-specific debuggers, such as pdb for Python.

- CNTK: With CNTK, there are three ways to visualize your neural network. The first way is to use the TensorBoard from TensorFlow. The second one is to use Graphviz. And the third one is to use the *visualize_network()* function to generate a PNG plot.

  In case of debugging, CNTK offers the *cntk.debugging* package, with which you can track every step of the training process and get an informative output. In addition, with the *performance profiler* users can debug performance issues. It generates a summary report and a detailed profile log. [13]

- Chainer: Chainer offers a possibility in case of visualization. It can generate a computational graph by invoking *build_computational_graph()*. The generated graph gets then dumped to the specific format dot.

  For debugging, Chainer has a mode called *Debug Mode*. There Chainer verifies the values of variables during the runtime and can give more detailed error messages. [11]

- PyTorch: Like CNTK, PyTorch can use TensorBoard to visualize models and data. [4] Another option is to use the small package *PyTorchViz* to create visualizations of execution graphs. In terms of debugging, PyTorch does only provide an API for debugging bottlenecks. For further debugging, developer must fall back on the debugging feature of Python or of the development tool (like PyCharm).

The clear winner in terms of visualization and debugging is TensorFlow. Only TensorFlow offers a board for visualization and debugging, which can also be used by other frameworks.

E. Languages supported

- TensorFlow: Tensorflow support most of the languages in our comparison. It supports Python, JavaScript, C++, Java, Go and Swift. There are also developments (by Google and its community) to support C# (TensorFlowSharp and TensorFlow.NET), Haskell, Julia, MATLAB, R, Ruby, Rust and Scala. [3]

- MXNet: MXNET offers support for Python, Scala, Julia, Clojure, Java, C++, R and Perl. [5]

- Chainer: Here we have the least number of supported languages. Chainer only supports Python. [11]

- CNTK: CNTK provides support for Python, C++, Java (currently experimental) and C#/.NET. [13]

- PyTorch: PyTorch supports Python and C++. [4]

We can also see that Python is supported by all frameworks. This is understandable as Python enjoys great popularity in the deep learning community. TensorFlow is dominating this part. It currently supports six languages and is working on eight more.

F. Distributed training

- TensorFlow: Distributed training over multiple GPUs, multiple machines or TPUs can be done under the *tf.distrbuted.Strategy* API. There are several strategies shown, which cover different use cases. There are six strategies available: MirroredStrategy, TPUStrategy, MultiWorkerMirroredStrategy, CentralStorageStrategy and ParameterServerStrategy. With those, there is a good performance out of the box, and it is possible to switch easily between strategies. [4]

- MXNet: MXNet has two approaches in distributed training: data parallelism and model parallelism. With data parallelism, each device stores a copy of the model. Then the dataset gets split up and the devices train on their part of the dataset. After that, they update collectively a shared model.

  The model parallelism approach comes into consideration when the models are too large to fit into a device memory. There are different devices assigned to different tasks of the model. Currently, MXNet only support this approach on a single machine. [5]

- CNTK: CNTK supports three methods of distributed training: data-parallel training, 1-bit SGD and BlockMomentum. With data-parallel training each minibatch of is going to be distributed over N worker processes with each one with one GPU. Afterwards, the gradients from the workers are collected before updating each model copy. In 1-bit SGD, there are techniques to speed up the communication part in the data-parallel training. For BlockMomentum, there are techniques used to improve the bandwidth by exchanging gradients every N minibatches. The communication is done by MPI (Message Passing Interface). [14]

- Chainer: Chainer provides inner-node training (multiple GPUs inside a node) and inter-node training (multiple nodes with one GPU). It is scalable with NVIDIA NCLL and CUD-Aware MPI for Communication. In addition, there are only minimal changes required to existing user code. [8] This all gets provided through the ChainerMN package and the MPI.

  Like the MXNet, it has two approaches for distributed training: the data parallel approach and the model parallel approach. [11] The model parallel approach API is still experimental.

- PyTorch: There are three main components in the PyTorch approach for distributed training: *Distributed Data-Parallel Training* (DDP), *RPC-Based Distributed Training* (RPC) and *Collective Communication.* DPP makes the single-program multiple-data approach like with the rest of the frameworks possible. It takes care of the model replicas synchronization and of the gradient communications. RPC supports the training structures, like distributed pipeline parallelism or parameter server paradigm, that are not able to fit the data-parallel training. It manages also remote object lifetime. With the Collective Communication, we can send tensors across processes inside a group. It offers collective communication APIs and P2P communication APIs. [4]

All the frameworks offer model parallelization and data parallelization. In case to cover the most use cases, TensorFlow is the right way. If your workload contains a node with several GPU, then the choice is Chainer.

G. GPU supported

- TensorFlow: TensorFlow provides support for NVIDIA GPUs natively through CUDA [3] and for AMD GPUs through the new open framework ROCm made by AMD. [15]

- PyTorch: Like TensorFlow, PyTorch support NVIDIA and AMD GPUs through CUDA and ROCm. [4][15]

- MXNet, CNTK and Chainer currently only support the GPUs from NVIDIA through CUDA. [5] [6] [11]

None of our frameworks supports the dedicated GPUs from Intel. It is also important to note, that the support for AMD GPUs by PyTorch and TensorFlow are only possible through the ROCm framework and not natively.

H. Deployment

- TensorFlow: TensorFlow offers *TensorFlow Extended (TFX)* as an end-to-end platform to deploy your model from research and development to production. It can create and manage pipelines, which are a sequence of components that implement a pipeline which are designed for scalable, high-performance tasks. [16] There is TensorFlow Serving, which is a high-performance, flexible serving system for models designed for production environment. It is included in the TFX and offers a Client API and Server API. [16]

  With TFLite, TensorFlow is capable for on-device inference. It supports Android, iOS, embedded Linux, and Microcontrollers. [3] TFX provides native support for TFLite to make inference on mobile devices possible.

- MXNet: To deploy your trained model in MXNet, you can export it. To export the model for inference applications, there is four different ways to deploy it: deploy to mobile devices through Amalgamation, to IoT devices with AWS Greengrass, to serverless application with AWS Lambda or to deploy it to a container environment (like Docker). [5][2]

- CNTK: With CNTK, there is several functionalities, with which you can deploy the model. Through ASP.NET, you can make your model accessible locally for inference. Another way is to deploy the model to Microsoft's cloud service Azure. There the model can be made accessible through a WebAPI or through the Azure Container Service with Kubernetes. [6]

- Chainer: Chainer is officially supported by Amazon Web Service (AWS). So, it is possible to train a deep learning model directly in the AWS environment and make it accessible for other AWS applications. There is also the possibility to deploy a model to a container environment (like Docker) or use it locally for inference. [8]

- PyTorch: There are several serving strategies for the deployment of a model. The first is direct embedding. Here we can embed the model into a mobile device or an embedded system (e.g., robot), in case that there is a Python or C++ environment support. An alternative for mobile devices is to use the ONNX export functionality and include it another framework. The second strategy is the model microservice. With this,

the model can be packed into a mechanism like a Docker container. So, we can handle and manage several models separately. The third way is to include the model into a Flask (a Python Web framework) and expose the model through a REST API for inference. There are also possibilities to include a model from PyTorch into MLFlow, Kubeflow, and RedisAI. The last approach is to use the cloud to deploy the model. Here we have the choice between the AWS Sagemaker with its Python SDK, the Google Cloud Platform Deep Learning VM with PyTorch pre-installed, the Alibaba Cloud, and the Microsoft Azure Machine Learning Service. [4]

It is also to be noted, that the current biggest cloud providers AWS, Google Cloud Platform and Microsoft Azure all provide functionalities to deploy a model from all the presented frameworks. In addition, there is also the possibility to deploy a model through an export to an ONNX format, which is supported by all frameworks.

All frameworks offer possibilities to deploy a model to a production environment. We can see, that TensorFlow, MXNet and PyTorch also offer ways to deploy to a mobile device or an embedded device (for example a robot). So, they provide the most production environments. However, the choice of deployment depends on the specific production environment.

In terms of distributed training, all of them offer this important functionality. It depends on the workload and the tasks for a clear choice. Therefore, we cannot name a clear winner for all use cases and workloads.

When it comes to the GPU-support, they all support NVIDIA-GPU. Only PyTorch and TensorFlow also support AMD-GPU through the ROCm framework.

Finally, for the deployment criteria, we have two framework which offers similar specs. PyTorch and TensorFlow offer deployment to the cloud, to mobile and embedded devices.

At the end, it can be observed that TensorFlow has a broad coverage on the APIs, the lead in visualization, debugging, languages and GPU-Support and covers a lot of use cases for distributed training and deployment part. It emerged as the winner in most points from the comparison. Therefore, it is the winner of our comparison.

Nevertheless, the choice can be different if the use case, the workload, or the hardware require a different framework. There is no clear winner for all use cases, as the frameworks also have different performance for different use cases. [2]

Framework	Documentation	API-Availability	ONNX	Visualization /Debugging	Languages	Distributed training	GPU-Support	Deployment
TensorFlow	Great	Great	tf2onnx	TensorBoard	14	Yes	NVIDIA/AMD	server/mobile/ cloud
MXNet	Good	Good	Native	Computational Graph/gdb	8	Yes	NVIDIA	cloud/mobile/ IoT
CNTK	Good	Good	Native	TensorBoard/Graphviz	4	Yes	NVIDIA	Azure/ASP.NET
Chainer	Good	Great	Native	dot/Debug Mode	1	Yes	NVIDIA	cloud
PyTorch	Great	Great	Native	TensorBoard/ -	2	Yes	NVIDIA/AMD	server/mobile/ cloud

Table 2: Framework Criteria Comparison

V. Conclusion

In this work, we first compared the documentation of the deep learning frameworks. All of them have good documentation but TensorFlow and PyTorch are slightly more detailed and structured. However, the documentation of them is sufficient.

Next the API-Availability was compared. There it depends on the use case. To cover the most environments, TensorFlow is your best choice and to get started with a basic pre-trained model, PyTorch is the best way with its hub.

For the ONNX-Support, all our compared frameworks supported the standard. They all support it natively expect for TensorFlow, where we have the *tf2onnx* converter as a solution.

In the case of visualization and debugging, TensorFlow is clearly in the lead. With its own TensorBoard, it offers the best visualization and debugging features. PyTorch did the worst, as it did not have an option of its own for debugging.

Most of the languages supported can be found in TensorFlow, in total fourteen.

References

[1] J. Liu, J. Dutta, N. Li, U. Kurup, and M. Shah. 2018. *Usability Study of Distributed Deep Learning Frameworks For Convolutional Neural Networks.* In 24th ACM SIGKDD conference on knowledge discovery and data minig.

[2] G. Nguyen, S. Dlugolinsky, M. Bobak, V. Tran, A. L. Garcia, I. Heredia, P. Malik, and L. Hluchy. 2019. *Machine Learning and Deep Learning frameworks and libraries for large scale data mining: a survey*. Springer.

[3] TensorFlow Homepage. Available: https://www.tensorflow.org (Accessed on 10.12.2020)

[4] PyTorch Homepage. Available: https://pytorch.org/ (Accessed on 10.12.2020)

[5] MXNet Homepage. Available: https://mxnet.apache.org/versions/1.7.0/ (Accessed on 10.12.2020)

[6] The Microsoft Cognitive Toolkit. Available: https://docs.microsoft.com/en-us/cognitive-toolkit/ (Accessed on 10.12.2020)

[7] Caffe Homepage. Available: https://caffe.berkeleyvision.org/ (Accessed on 10.12.2020)
[8] Chainer Homepage. Available: https://chainer.org/ (Accessed on 10.12.2020)
[9] Deeplearning4j Homepage. Available: https://deeplearning4j.org/ (Accessed on 10.12.2020)
[10] PC graphics processing nit shipment share worldwide. Available: https://www.statista.com/statistics/754557/worldwide-gpu-shipments-market-share-by-vendor (Accessed on 10.12.2020)
[11] Chainer Documentation. Available: https://docs.chainer.org/en/stable/reference/index.html (Accessed on 10.12.2020)
[12] Caffe2 Blog. Available: https://caffe2.ai/blog/2018/05/02/Caffe2_PyTorch_1_0.html (Accessed on 10.12.2020)
[13] CNTK Library API. Available: https://docs.microsoft.com/en-us/cognitive-toolkit/cntk-library-api (Accessed on 10.12.2020)
[14] API for CNTK. Available: https://www.cntk.ai/pythondocs/CNTK_200_GuidedTour.html (Accessed on 10.12.2020)
[15] ROCm: machine learning. Available: https://www.amd.com/de/graphics/servers-solutions-rocm-ml (Accessed on 10.12.2020)
[16] TensorFlow Extended. Available: https://www.tensorflow.org/tfx (Accessed on 10.12.2020)
[17] S. Shi, Q. Wang, and X. Chu. 2018. *Performance Modeling and Evaluation of Distributed Deep Learning Frameworks on GPUs.* In IEEE International Symposium on Dependable, Autonomic and Secure Computing (DASC)
[18] R. D. Fonnegra, B. Blair, and G. M. Diaz. 2017. *Performance comparison of deep learning frameworks in image classification problems using convolutional and recurrent networks.* In IEEE Colombian Communications Conference (COLCOM)
[19] ONNX Supported Tools. Available: https://onnx.ai/supported-tools.html (Accessed on 10.12.2020)
[20] Github Homepage. Available: https://github.com/ (Accessed on 30.12.2020)