Chapter 10
Resources

Abstract Deep learning has been shown as a powerful method for a variety of artificial intelligence tasks, including some critical tasks in NLP. However, training a deep neural network is usually a very time-intensive process and requires lots of code to build related models. To alleviate these issues, some deep learning frameworks have been developed and released, which incorporate some existing and necessary arithmetic operators for neural network constructions. And these frameworks exploit hardware features such as multi-core CPUs and many-core GPUs to shorten the training time. Each framework has its advantages and disadvantages. In this chapter, we aim to exhibit features and running performance of these frameworks so that users can select an appropriate framework for their usage.

10.1 Open-Source Frameworks for Deep Learning

In this section, we will introduce several typical open-source frameworks for deep learning including Caffe, Theano, TensorFlow, Torch, PyTorch, Keras, and MXNet. In fact, as the rapid development of the deep learning community, these open-source frameworks are updating every day, and therefore the information in this section may not be up to date. In fact, this section mainly focuses on introducing the special features of these frameworks and lets the readers have a preliminary understanding of them. To know the latest features of these deep learning frameworks, please refer to their official sites.

10.1.1 Caffe

Caffe[1] is a well-known framework and is widely used for computer vision tasks. It was created by Yangqing Jia and developed by Berkeley AI Research (BAIR). Caffe uses a layer-wise approach to make building models become easy, and it is also

[1] http://caffe.berkeleyvision.org/.

Z. Liu et al., *Representation Learning for Natural Language Processing*,
https://doi.org/10.1007/978-981-15-5573-2_10

convenient to fine-tune the existing neural networks without writing too much code via its simple interfaces. The underlying designs of Caffe are for the fast construction of convolutional neural networks, which make it efficient and effective.

On the other hand, as normal pictures often have a fixed size, the interfaces of Caffe are fixed and hard to be extended. It is thus difficult to use Caffe for other tasks with a variable input length, such as text, sound, or other time-series data. Recurrent neural networks are also not well supported by Caffe. Although users can easily build an existing network architecture with the layer-wise framework, it is not flexible when dealing with big and complex networks. If users want to design a new layer, the users need to use C/C++ and CUDA for the underlying coding of the new layer.

10.1.2 Theano

Theano[2] is the typical framework developed to use symbolic tensor graphs for model specification. Any neural networks or other machine-learning models can be represented as symbolic tensor graphs. Forward, backward, and gradient updates can be calculated based on the flow between tensors. Hence, Theano provides more flexibility than Caffe using a layer-wise approach to build models. In Caffe, to define a new layer that is not already in the existing repository of layers is complicated, which needs to implement its forward, backward, and gradient update functions before. In Theano, you only need to use basic operators to define the customized layer following the order of operations.

Theano is a platform and is easy to configure as compared with other frameworks. And some high-level frameworks are built on top of Theano such as Keras, which further makes Theano easier to use. Theano supports cross-platform configuration well, which means it works on not only Linux but also Windows. Because of this, many researchers and engineers use Theano to build their models and then release these projects. Rich open resources based on Theano attract some more users.

Though Theano uses Python syntax to define symbolic tensor graphs, its graph processor will compile the graphs into high-performance C++ or CUDA code for computing. Owing to this, Theano can run very fast and make programmers code mode simply. Only one deficiency is that the compilation process is slow and needs some time. If a neural network does not need to be trained for several days, it is not a good idea to select Theano. Compiling too often in Theano is maddening and annoying. As a comparison, the later framework like TensorFlow uses the compiled package for the symbolic tensor operations, which seems a little more relaxing.

Theano has some other serious disadvantages. Theano cannot support many-core GPUs very well, which makes it hard to train big neural networks. Besides the compilation process, importing Theano is also slow. When you run your code in Theano, you will be stuck for a long time with a preconfigured device. If you want to improve and contribute to Theano itself, this will also be maddening and annoying.

[2] http://www.deeplearning.net/software/theano/.

In fact, Theano is no longer maintained, but it is still worth introducing as a landmark work in the history of deep learning frameworks, which inspires many subsequent frameworks.

10.1.3 TensorFlow

TensorFlow[3] is mainly developed and used by Google based on the experience on Theano and DistBelief [1]. TensorFlow and Theano are in fact quite similar to some extent. Both of them allow building a symbolic graph of the neural network architecture via the Python interface. Different from Theano, TensorFlow allows implementing new operations or machine-learning algorithms using C/C++ and Java. With building symbolic graphs, the auto-gradient can be easily used to train complicated models. Hence, TensorFlow is more than a deep learning framework. Its flexibility enables it to solve various complex computing problems such as reinforcement learning.

In TensorFlow, both code development and deployment is fast and convenient. Trained models can be deployed quickly on a variety of devices, including servers and mobile devices, without the need to implement a separate model setting code or load Python/LuaJIT interpreter. Caffe also allows easy deployment of models. However, Caffe has trouble running on devices without a GPU, which is a prevalent situation of smartphones. TensorFlow supports model decoding using ARM/NEON instructions and does not need too many operations to choose training devices.

TensorBoard of TensorFlow provides a platform for visualization of the model architectures, which is beautiful and also useful. By visualizing the symbolic graph, it is not difficult to find bugs in the source code. To debug models on other deep learning frameworks is relatively bothering. TensorBoard can also log and generate real-time visualization of variables during training, which is a pleasant way to monitor the training process.

Though customizing operations in TensorFlow is convenient, it usually changes a lot of function interfaces in every new release which is challenging for developers to keep their code compatible with different TensorFlow versions. And mastering TensorFlow is also not easy. As TensorFlow 2.0 has been released recently, TensorFlow may gradually handle these issues in the predictable future.

10.1.4 Torch

Torch[4] is a computational framework mainly developed and used by Facebook and Twitter. Torch provides an API written in Lua to support the implementation of some

[3] https://www.tensorflow.org/.

[4] http://torch.ch/.

machine-learning algorithms, especially convolutional neural networks. A temporal convolutional layer implemented by Torch can have a variable input length, which is extremely useful for NLP tasks and not designed in Theano and TensorFlow. Torch also contains the 3D convolutional layer, which can be easily used in video recognition tasks. Besides its various flexible convolutional layers, Torch is light and speedy. The above reasons attract lots of researchers in universities and companies to customize their own deep learning platforms.

However, the negative aspects of Torch are also apparent. Though Torch is powerful, it is not designed to be widely accessible to the Python-based academic community. And there are not any other interfaces but Lua. Lua is a multi-paradigm scripting language, which was developed in Brazil in the early 1990s and is not a popular mainstream programming language. Hence, it needs some time to learn Lua before you use Torch to construct models. Different from convolutional neural networks, there is no official support for recurrent neural networks. There are some open resources about recurrent neural networks implemented by Torch, but they are not yet integrated to the main repository. And it is difficult to distinguish the effectiveness of these implementations.

Similar to Caffe, Torch is not a framework based on symbolic tensor graphs, it also uses the layer-wise approach. This means that your models in Torch are a graph of layers and not a graph of mathematical functions. The mechanism is convenient to build a network whose layers are stable and hierarchical. If you want to design a new connection layer or change an existing neural model, you need lots of code to implement new layers with full forward, backward, and gradient update functions. However, those frameworks based on symbolic tensor graphs, such as Theano and TensorFlow, give more flexibility to do this. In fact, these issues are handled as PyTorch has been released, which we will introduce then.

10.1.5 PyTorch

PyTorch[5] is a Python package built over Torch, developed by Facebook and other companies. However, it is not just an interface, and PyTorch has amounts of improvements over Torch. The most important one is that PyTorch can use a symbolic graph to define neural networks, and then use automatic differentiation following the graph to automate the computation of backward passes in neural networks. Meanwhile, PyTorch maintains some characteristics of the layer-wise approach in Torch, which means coding with PyTorch is easy. Moreover, PyTorch has minimal framework overhead and custom memory allocators for the GPU, which means PyTorch is faster and memory-efficient than Torch.

[5] http://pytorch.org/.

Compared with other deep learning frameworks, PyTorch has two main advantages. First, most frameworks like TensorFlow are based on static computational graphs (define-and-run), while PyTorch uses dynamic computational graphs (define-by-run). What it means is that with dynamic computational graphs, you can change the network architecture based on the data flowing through the network. There is a way to do something similar in TensorFlow, but your static computational graphs must contain all possible branches in advance, which will limit the performance. Second, PyTorch is built to be deeply integrated into Python and has a seamless connection with other popular Python packages, such as Numpy, Spicy, and Cython. Thus, it is easy to extend your model when needed.

After Facebook released it, PyTorch has drawn considerable attention from the deep learning community, and many past Torch users switch to this new package. For now, PyTorch already has a thriving community which contributes to its increasing popularity among researchers. It is no exaggeration to say that PyTorch is one of the most popular frameworks at present.

10.1.6 Keras

Keras[6] is a top-design deep learning framework that is based on Theano and TensorFlow. Interestingly, Keras sits atop Theano and TensorFlow, however, its interfaces are similar to Torch. To use Keras needs Python code, and there are lots of detailed documents and examples for a quick start. There is also a very active community of developers, and they make Keras fastly updated. Hence, it is a very fast-growing framework.

Because Theano and TensorFlow are the backends of Keras, disadvantages of Keras are most similar to Theano and TensorFlow. With TensorFlow as the backend, it will run even slower than the pure TensorFlow code. Because it is a high-level framework, to customize a new neural layer is not easy, though you can easily use existing layers under Keras. The package is too advanced, and it hides too many training parameters. You cannot touch and change all details of your own models unless you use Theano, TensorFlow, or PyTorch.

10.1.7 MXNet

MXNet[7] is an effective and efficient open-source machine-learning framework, mainly pushed by Amazon. It supports APIs with multiple languages, including C++, Python, R, Scala, Julia, Perl, MATLAB, and JavaScript, some of which can be adopted for Amazon Web Services. Some interfaces of MXNet are also reserved for

[6] https://keras.io/.

[7] http://mxnet.io/.

future mobile devices, just like TensorFlow. MXNet is built on a dynamic dependency scheduler that automatically parallelizes both symbolic and imperative operations on the fly. A graph optimization layer on top of that makes symbolic execution fast and memory efficient. The MXNet library is portable and lightweight, and it scales to multiple GPUs and multiple machines. The main problem of MXNet is the lack of detailed and well-organized documentation. The user groups are also smaller than other frameworks, especially as compared with TensorFlow and PyTorch. It is more challenging to grasp MXNet for newbies. The MXNet is developing fastly, and these problems may be solved in the future.

10.2 Open Resources for Word Representation

10.2.1 Word2Vec

Word2vec[8] is a widely used toolkit for word representation learning, which provides an effective and efficient implementation of the continuous bag-of-words and Skip-gram architectures. The word representations learned by Word2vec can be used in many natural language processing fields. Empirically, To use pretrained word vectors as the model inputs can be a good way to enhance model performances.

Word2vec takes free text corpus as input and constructs the vocabulary list from the training data. Then it uses simple predictive models based on neural networks to learn the language model, which encode the co-occurrence information between words into the resulting word representations.

The resulting representations showcase interesting linear substructures of the word vector space. The Euclidean distance (or cosine similarity) between two-word vectors provides an effective method for measuring the linguistic or semantic similarity of the corresponding words. Sometimes, the nearest neighbors, according to this metric, reveal rare but relevant words that lie outside an average human's vocabulary.

Words frequently appearing together in the text will have representations with close distance within the embedding space. Word2vec also provides a tool to find the closest words for a user-specified word via the learned representations and distances between representation embeddings.

10.2.2 GloVe

GloVe[9] is a widely used toolkit, which supports an unsupervised learning method for word representation learning. Similar to Word2vec, GloVe also trains on text corpus

[8] https://code.google.com/archive/p/word2vec/.

[9] https://nlp.stanford.edu/projects/glove/.

and captures the aggregated global word-word co-occurrence information for word embeddings. However, GloVe uses count-based models instead of predictive models, which are different from Word2vec.

The GloVe model first builds a global word-word co-occurrence matrix, which can show how frequently words co-occur with one another in a given text. Then word representations are trained on the nonzero entries of the matrix. To construct this matrix requires the entire corpus traversal for the statistics collection. For large corpora, this pass can be computationally expensive, but it is a one-time up-front cost. Subsequent training iterations are much faster because the number of nonzero matrix entries is typically much smaller than the total number of words in the corpus.

10.3 Open Resources for Knowledge Graph Representation

10.3.1 OpenKE

OpenKE[10] [2] is an open-source toolkit for Knowledge Embedding (KE), which provides a unified framework and various fundamental KE models. OpenKE prioritizes operational efficiency to support quick model validation and large-scale knowledge representation learning. Meanwhile, OpenKE maintains sufficient modularity and extensibility to incorporate new models easily. Besides the toolkit, the embeddings of some existing large-scale knowledge graphs pretrained by OpenKE are also available. The toolkit, documentation, and pretrained embeddings are all released on http://openke.thunlp.org/.

As compared to other implementations, OpenKE has five advantages. First, OpenKE has implemented nine classical knowledge embedding algorithms, including RESCAL, TransE, TransH, TransR, TransD, ComplEx, DistMult, HolE, and Analogy, which are verified effective and stable. Second, OpenKE shows high performance due to memory optimization, multi-threading acceleration, and GPU learning. OpenKE supports multiple computing devices and provides interfaces to control CPU/GPU modes. Third, system encapsulation makes OpenKE easy to train and test KE models. Users just need to set hyperparameters via interfaces of the platform to construct KE models. Fourth, it is easy to construct new KE models. All specific models are implemented by inheriting the base class by designing their own scoring functions and loss functions. Fifth, besides the toolkit, OpenKE also provides the embeddings of some existing large-scale knowledge graphs pretrained by OpenKE, which can be directly applied for many applications, including information retrieval, personalized recommendation, and question answering.

[10] https://github.com/thunlp/OpenKE.

10.3.2 Scikit-Kge

Scikit-kge[11] is an open-source Python library for knowledge representation learning. The library supports different building blocks to train and develop models for knowledge graph embeddings. The primary purpose of Scikit-kge is to compute the embeddings of knowledge graphs for the method HolE; meanwhile, it also provides some other methods. Besides HolE, RESCAL, TransE, TransR, and ER-MLP can also be trained in Scikit-kge. The library contains some parameter update methods, not only the basic SGD but also AdaGrad. It also implements different negative sampling strategies to select negative samples.

10.4 Open Resources for Network Representation

10.4.1 OpenNE

OpenNE[12] is an open-source standard NE/NRL (Network Representation Learning) training and testing framework. It unifies the input and output interfaces of different NE models and provides scalable options for each model. Moreover, typical NE models under this framework are based on TensorFlow, which enables these models to be trained with GPUs. The implemented or modified models include DeepWalk, LINE, node2vec, GraRep, TADW, GCN, HOPE, GF, SDNE, and LE. The framework also provides classification and embedding visualization modules for evaluating the result of NRL.

10.4.2 GEM

GEM (Graph Embedding Methods)[13] is a Python package that offers a general framework for graph embedding methods. It implements many state-of-the-art embedding techniques including Locally Linear Embedding, Laplacian Eigenmaps, Graph Factorization, High-Order Proximity preserved Embedding (HOPE), Structural Deep Network Embedding (SDNE), and node2vec. Furthermore, the framework implements several functions to evaluate the quality of the obtained embeddings including graph reconstruction, link prediction, visualization, and node classification. For faster execution, C++ backend is integrated using Boost for supported methods.

[11] https://github.com/mnick/scikit-kge.
[12] https://github.com/thunlp/OpenNE.
[13] https://github.com/palash1992/GEM.

10.4.3 GraphVite

GraphVite[14] is a general and high-performance graph embedding system for various applications including node embedding, knowledge graph embedding, and graph high-dimensional data visualization.

GraphVite provides a complete pipeline for users to implement and evaluate graph embedding models. For reproducibility, the system integrates several commonly used models and benchmarks, and you can also develop your own models with the flexible interface. Additionally, for semantic tasks, GraphVite releases a bunch of pretrained knowledge graph embedding models to enhance language understanding. There are two core advantages of GraphVite over other toolkits: fast and large-scale training. GraphVite accelerates graph embedding with multiple CPUs and GPUs. It takes around one minute to learn node embeddings for graphs with one million nodes. Moreover, GraphVite is designed to be scalable. Even with limited memory, GraphVite can process node embedding task on billion-scale graphs.

10.4.4 CogDL

CogDL[15] is another graph representation learning toolkit that allows researchers and developers to easily train and evaluate baseline or custom models for node classification, link prediction, and other tasks on graphs. It provides implementations of many popular models, including non-GNN models and GNN-based ones.

CogDL benefits from several unique techniques. First, utilizing sparse matrix operation, CogDL is capable of performing fast network embedding on large-scale networks. Second, CogDL has the ability to deal with different types of graph structures attributed, multiplex, and heterogeneous networks. Third, CogDL supports parallel training. With different seeds and different models, CogDL performs training on multiple GPUs and reports the result table automatically. Finally, CogDL is extendable. New datasets, models, and tasks can be added without difficulty.

10.5 Open Resources for Relation Extraction

10.5.1 OpenNRE

OpenNRE[16] [3] is an open-source framework for neural relation extraction, which aims to easily build relation extraction (RE) models.

[14] https://graphvite.io/.

[15] http://keg.cs.tsinghua.edu.cn/cogdl/index.html.

[16] https://github.com/thunlp/OpenNRE.

Compared with other implementations, OpenNRE has four advantages. First, OpenNRE has implemented various state-of-the-art RE models, including attention mechanism, adversarial learning, and reinforcement learning. Second, OpenNRE enjoys great system encapsulation. It divides the pipeline of relation extraction into four parts, namely, embedding, encoder, selector (for distant supervision), and classifier. For each part, it has implemented several methods. System encapsulation makes it easy to train and test models by changing hyperparameters or appoint model architectures by using Python arguments. Third, OpenNRE is extendable. Users can construct new RE models by choosing specific blocks provided in four parts as mentioned above and combining them freely, with only a few lines of codes. Fourth, the framework has implemented multi-GPU learning, which is efficient.

References

1. Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.
2. Xu Han, Shulin Cao, Xin Lv, Yankai Lin, Zhiyuan Liu, Maosong Sun, and Juanzi Li. OpenKE: An open toolkit for knowledge embedding. In *Proceedings of EMNLP: System Demonstrations*, pages 139–144, 2018.
3. Xu Han, Tianyu Gao, Yuan Yao, Deming Ye, Zhiyuan Liu, and Maosong Sun. OpenNRE: An open and extensible toolkit for neural relation extraction. In *Proceedings of EMNLP: System Demonstrations*, pages 169–174, 2019.