Graph neural networks in TensorFlow-Keras with RaggedTensor representation (kgcnn)

Patrick Reiser $^{a,b,*}$, André Eberhard $^b$, Pascal Friederich $^{a,b,**}$

$^a$ Institute of Nanotechnology, Karlsruhe Institute of Technology (KIT), Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany
$^b$ Institute of Theoretical Informatics, Karlsruhe Institute of Technology (KIT), Am Fasanengarten 5, 76131 Karlsruhe, Germany

** ABSTRACT **

Graph neural networks are a versatile machine learning architecture that received a lot of attention recently due to its wide range of applications. In this technical report, we present an implementation of graph convolution and graph pooling layers for TensorFlow/Keras models, which allows a seamless and flexible integration into standard Keras layers to set up graph models in a functional way. We developed the Keras Graph Convolutional Neural Network Python package *kgcnn* based on TensorFlow-Keras which focus on a transparent tensor structure passed between layers and an ease-of-use mindset.

---

** Code metadata **

- Current code version: v1.0.0
- Permanent link to code/repository used for this code version: https://github.com/SoftwareImpacts/SIMPAC-2021-57
- Permanent link to reproducible capsule: https://codeocean.com/capsule/8499626/tree/v1
- Legal code licence: MIT Licence
- Code versioning system used: Git
- Software code languages, tools, and services used:
  - Python 3
  - TensorFlow $\geq 2.4$
- Compilation requirements, operating environments & dependencies:
  - Python 3
  - TensorFlow $\geq 2.4$
- Support email for questions: patrick.reiser@kit.edu

** Software metadata **

- Current software version: v1.0.0
- Permanent link to executables of this version: https://pypi.org/project/kgcnn/
- Permanent link to reproducible capsule: https://codeocean.com/capsule/8499626/tree/v1
- Legal software licence: MIT Licence
- Computing platforms/Operating systems:
  - Linux
  - Microsoft Windows
  - Mac OS
- Installation requirements & dependencies:
  - TensorFlow $\geq 2.4$
- If available, link to user manual - if formally published include a reference to the publication in the reference list
- Support email for questions: patrick.reiser@kit.edu

The code (and data) in this article has been certified as Reproducible by Code Ocean: [https://codeocean.com/](https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at [https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals](https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals).

* * Corresponding author at: Institute of Nanotechnology, Karlsruhe Institute of Technology (KIT), Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany.
** Corresponding author at: Institute of Theoretical Informatics, Karlsruhe Institute of Technology (KIT), Am Fasanengarten 5, 76131 Karlsruhe, Germany.

E-mail addresses: patrick.reiser@kit.edu (P. Reiser), pascal.friederich@kit.edu (P. Friederich).

https://doi.org/10.1016/j.simpa.2021.100095

Received 25 May 2021; Received in revised form 1 June 2021; Accepted 2 June 2021

2665-9638/© 2021 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).
1. Introduction

Graph neural networks (GNNs) are a natural extension of common neural network architectures such as convolutional neural networks (CNN) [1–3] for image classification to graph structured data [4]. For example, recurrent [5,6], convolutional [4,7–9] and spatial–temporal [10] graph neural networks as well as graph autoencoders [11,12] and graph transformer models [13,14] have been reported in literature. A graph \( G = (V,E) \) is defined as a set of vertices or nodes \( v_i \in V \) and edges \( e_{ij} = (v_i,v_j) \in E \) connecting two nodes. There are already comprehensive and extensive review articles for graph neural networks, which summarize and categorize relevant literature on graph learning [15,16]. The most frequent applications of GNNs include node classification or graph embedding tasks. While node classification is a common task for very large graphs such as citation networks [12] or social graphs [17], graph embeddings learn a representation of smaller graphs such as molecules [7,18] or text classifications [19,20]. Early work on applying neural networks to graphs [21] shaped the notion of graph neural networks and was further elaborated on by propagating information iteratively through the graph [5,18,22,23]. Graph networks may operate on the graph’s spectrum [24,25] or directly on its structure [26]. One of the most prominent graph convolutional neural network (GCN) introduced by Kipf et al. [4] stacks multiple convolutional and pooling layers for deep learning to generate a high-level node representation from which both a local and global graph classification can be obtained. Most GCNs can be considered as message passing networks [27,28], a class of graph networks in which information is propagated along edges between neighbouring nodes. In each update step \( t \), the central node’s hidden representation \( h_v \) is convolved with its neighbourhood given by:

\[
h_v^{t+1} = U_t(h_v^t, m_v^t),
\]

where \( m_v^t \) denotes the aggregated message and \( U_t \) the update function. The message to update is usually acquired from summing message functions \( M_t \) from the neighbourhood \( N(v) = \{ u \in V | (u,v) \in E \} \) of node \( v \):

\[
m_v^{t+1} = \sum_{u \in N(v)} M_t(h_v^t, h_u^t, e_{uv}).
\]

More complex aggregation schemes are of course possible [29]. There is a large variety in convolution operators, which can be spectral-based [30] or spatial-based involving direct neighbours or a path of connected nodes to walk and collect information from [14,28]. Moreover, the message and update functions can be built from recurrent networks [31], multi-layer perceptrons (MLP) [32] or attention heads [33] which are complemented by a set of possible aggregation or pooling operations. Aggregation is usually done by a simple average of node representations or by a more refined set2set encoder part [34] as proposed by Gilmer et al. [27]. A reduction of nodes in the graph is achieved by pooling similar to CNNs but which is much more challenging on arbitrarily structured data. Examples of possibly differentiable and learnable pooling filters introduced in literature are DiffPool [35], EdgePool [36], gPool [37] HGP-SL [35], SAGPool [38], iPool [39], EigenPool [40] and graph based clustering methods such as the Graculus algorithm [25,41–43].

In order to utilize the full scope of different graph operations for setting up a custom GNN model, a modular framework of convolution and pooling layers is necessary. We briefly summarize and discuss existing graph libraries and their code coverage. Then, a short overview of representing graphs in tensor form is given. Finally, we introduce our graph package kgcn for TensorFlow 2’s Keras API [44–46], which seamlessly integrates graph layers into the Keras [47] environment.

### 1.1. Graph libraries

Since graph neural networks require modified convolution and pooling operators, many Python packages for deep learning have emerged for either TensorFlow [44,45] or PyTorch [48] to work with graphs. We try to summarize the most notable ones without any claim that this list is complete.

- **PyTorch Geometric** [49]. A PyTorch based graph library which is probably the largest and most used graph learning Python package up to date. It implements a huge variety of different graph models and uses a disjoint graph representation to deal with batched graphs (graph representations are discussed in the Supplementary Information).
- **Deep Graph Library (DGL)** [50]. A graph model library with a flexible backend and a performance optimized implementation. It has its own graph data class with many loading options. Moreover, variants such as generative graph models [51], Capsule [52] and transformers [53] are included.
- **Spektral** [54]. A Keras [47] implementation of graph convolutional networks. Originally restricted to spectral graph filters [30], it now includes spatial convolution and pooling operations. The graph representation is made flexible by different graph modes detected by each layer.
- **StellarGraph** [55]. A Keras [47] implementation that implements a set of convolution layers and a few pooling layers plus a custom graph data format.

With PyTorch Geometric and DGL, there are already large graph libraries with a lot of contributors from both academics and industry. The focus of the graph package presented here is on a neat integration of graphs into the TensorFlow-Keras framework in the most straightforward way. Thereby, we hope to provide Keras graph layers which can be quickly rearranged, changed and extended to build custom graph models with little effort. This implementation is focused on the new TensorFlow’s RaggedTensor class which is most suited for flexible data structures such as graphs and natural language. The main field of applications targeted with this package is graph embedding tasks of e.g. molecules, materials and contextual or knowledge graph learning.

### 2. Description

A flexible and simple integration of graph operations into the TensorFlow-Keras framework can be achieved via ragged tensors. As mentioned above, ragged tensors are capable of efficiently representing graphs and have inherent access to various methods within TensorFlow (see Supplementary Information). For more sophisticated pooling algorithms which cannot be operated on batches, a parallelization of individual graphs within the batch could be achieved with the TensorFlow map functionality, although this is less efficient than vectorized operations and depends on implementation details.

We introduce a Python package kgcn\(^1\) that uses RaggedTensors, which are passed between graph layers for graph convolution and message passing models. We believe that the use of RaggedTensors

\(^1\) https://github.com/aimat-lab/gcnn_keras.

**Table 1**

Mean absolute validation error for single training on QM9 dataset for targets such as molecular orbital energies (HOMO, LUMO) and the energy gap \( E_g \) in eV using popular GNN architectures implemented in kgcn. No hyperparameter optimization or feature engineering was performed.

<table>
<thead>
<tr>
<th>Model</th>
<th>HOMO [eV]</th>
<th>HOMO [eV]</th>
<th>( E_g ) [eV]</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPNN [27]</td>
<td>0.061</td>
<td>0.047</td>
<td>0.083</td>
</tr>
<tr>
<td>Schnet [7]</td>
<td>0.044</td>
<td>0.038</td>
<td>0.067</td>
</tr>
<tr>
<td>MegNet [32]</td>
<td>0.045</td>
<td>0.037</td>
<td>0.066</td>
</tr>
</tbody>
</table>
allows a transparent and readable coding style, enables a seamless integration with many TensorFlow methods which are available for custom layers and makes it easy to debug code. We implemented a set of basic Keras layers for TensorFlow 2 from which many models reported in literature can be constructed. A simple code example is shown in Listing 1.

```
import tensorflow.keras as ks
from kgcnn.layers.gather import GatherNodes
from kgcnn.layers.keras import Dense, Concatenate  # ragged support
from kgcnn.layers.pooling import PoolingLocalMessages, PoolingNodes

n = ks.layers.Input(shape=(None, 3), name='node_input',
dtype='float32', ragged=True)
ei = ks.layers.Input(shape=(None, 2), names='edge_index_input',
dtype='int64', ragged=True)

node_messages = Dense(10, activation='relu')(n_in_out)
node_updates = PoolingLocalMessages()(node_messages, ei)
n_node_updates = Concatenate(axis=-1)(node_messages, node_updates)

g_embedd = Dense(1)(n_node_updates)
message_passing = ks.models.Model(inputs=[n, ei], outputs=g_embedd)
```

Listing 1: Python code example for setting up a simple one-layer message passing network in the functional API of TensorFlow-Keras for a graph embedding task. The size of the network and the architecture for this example is chosen arbitrarily.

The Python package implements the following architectures as examples: GCN [4], Interaction network [9], message passing [27], Schnet [7], MegNet [32], Unet [37], GNN Explainer [56], Graph-SAGE [29], GAT [33] and DimeNet++ [57]. The focus is set on graph embedding tasks, but also node and link classification tasks can be implemented using kgcnn. The models were tested with common benchmark datasets such as Cora [58], MUTAG [59] and QM9 [60]. Typical benchmark accuracies such as chemical accuracy on the QM9 dataset are achieved with the corresponding models implemented in kgcnn (see Table 1).

3. Software impact

Learning dedicated graph embeddings is of high interest in for example classification tasks and molecular property [61–64] and reaction predictions [65–68]. A differentiable and continuous graph convolution model could in principle replace empirical constructed force fields for molecular dynamics simulation (MD) when trained on quantum mechanical calculations [7,69–71]. This includes work of the authors [28, 72, 73]. In general, representation learning allows us to encode graph structured knowledge about interacting entities (i.e. nodes) and store it into a low-dimensional vector which can be further processed by machine learning (ML) models [74]. The graph models re-implemented in kgcnn are widely applicable and are well established in literature [7,27]. We believe that kgcnn can be used by researchers in other scientific fields to accommodate different graph learning tasks without detailed knowledge about graph representations or implementation details and by working with the well-known TensorFlow-Keras deep learning environment.

4. Limitations and future development

The current version of kgcnn is targeted for graph representation learning of small graphs such as molecules. Training on a single graph instance, e.g. a citation network which does not fit into memory is currently not explored. Although distribution strategies for training are already integrated into TensorFlow’s high-level Keras API, they have not yet been tested with the models provided by kgcnn. We plan to continue to extend the kgcnn library to incorporate new models, in particular pooling methods [35] and to improve functionality. A future goal is to provide extremely large graph neural networks for distributed training on an exhaustive dataset for applications in chemistry and materials science.

5. Conclusion

In summary, we discussed a way to integrate graph convolution models into the TensorFlow-Keras deep learning framework. Main focus of our kgcnn package is the transparency of the tensor representation and the seamless integration with other Keras models.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

P.F. acknowledges funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 795206.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.simpa.2020.100095.

References


