

NeuronBlocks: Building Your NLP DNN Models Like Playing Lego

Ming Gong[§] Linjun Shou[§] Wutao Lin[§] Zhijie Sang[§] Quanjia Yan[‡]
Ze Yang[§] Feixiang Cheng[§] Daxin Jiang[§]

[§] STCA NLP Group, Microsoft, Beijing, China

[‡] Research Center for Ubiquitous Computing Systems, ICT, CAS, Beijing, China

{migon, lisho, wutlin, zhsang, yaze, fecheng, djiang}@microsoft.com

yanquanjia17s@ict.ac.cn

Abstract

Deep Neural Networks (DNN) have been widely employed in industry to address various Natural Language Processing (NLP) tasks. However, many engineers find it a big overhead when they have to choose from multiple frameworks, compare different types of models, and understand various optimization mechanisms. An NLP toolkit for DNN models with both generality and flexibility can greatly improve the productivity of engineers by saving their learning cost and guiding them to find optimal solutions to their tasks. In this paper, we introduce NeuronBlocks^{1, 2}, a toolkit encapsulating a suite of neural network modules as building blocks to construct various DNN models with complex architecture. This toolkit empowers engineers to build, train, and test various NLP models through simple configuration of JSON files. The experiments on several NLP datasets such as GLUE, WikiQA and CoNLL-2003 demonstrate the effectiveness of NeuronBlocks.

1 Introduction

Deep Neural Networks (DNN) have been widely employed in industry for solving various Natural Language Processing (NLP) tasks, such as text classification, sequence labeling, question answering, etc. However, when engineers apply DNN models to address specific NLP tasks, they often face the following challenges.

- Multiple DNN frameworks, including TensorFlow, PyTorch, Keras, etc. It is a big overhead to learn how to program under the frameworks.
- Diverse and fast evolving DNN models, such as CNN, RNN, and Transformer. It takes big efforts to understand the intuition and maths behind these models.

¹Code: <https://github.com/Microsoft/NeuronBlocks>

²Demo: <https://youtu.be/x6cOpVS2cdo>

- Various regularization and optimization mechanisms. To tune model performance for both quality and efficiency, model developers have to gain experience in Dropout, Normalization, Mixed precision training, etc.
- Coding and debugging complexity. Programming under DNN frameworks requires developers to be familiar with the built-in packages and interfaces. It needs much expertise to develop, debug, and optimize code.
- Platform compatibility. It requires extra coding work to run on different platforms, such as Linux/Windows, GPU/CPU.

The above challenges often hinder the productivity of engineers, and result in less optimal solutions to their given tasks. This motivates us to develop an NLP toolkit for DNN models. Before designing this NLP toolkit, we conducted a survey among engineers and identified a spectrum of three typical personas.

- The first type of engineers prefer off-the-shelf networks. Given a specific task, they expect the toolkit to suggest several end-to-end network architectures, and then they simply focus on collecting the training data, and tuning the model parameters. They hope the whole process to be extremely agile and easy.
- The second type of engineers would like to build the networks by themselves. However, instead of writing each line of code from scratch, they hope the toolkit to provide a rich gallery of reusable modules as building blocks. Then they can compare various model architectures constructed by the building blocks.
- The last type of engineers are advanced users. They want to reuse most part of the existing networks, but for critical components, they would like to make innovations and create their own modules. They hope the toolkit to have

an open infrastructure, so that customized modules can be easily plugged in.

To satisfy the requirements of all the above three personas, the NLP toolkit has to be generic enough to cover as many tasks as possible. At the same time, it also needs to be flexible enough to allow alternative network architectures as well as customized modules. Therefore, we analyzed the NLP jobs submitted to a commercial centralized GPU cluster. Table 1 showed that about 87.5% NLP related jobs belong to a few common tasks, including sentence classification, text matching, sequence labeling, machine reading comprehension (MRC), etc. It further suggested that more than 90% of the networks were composed of several common components, such as embedding, CNN/RNN, Transformer and so on.

Tasks	Ratio
Text matching	39.4%
Sentence classification	27.3%
Sequence labeling	14.7%
MRC	6.0%
Others	12.5%

Table 1: Task analysis of NLP DNN jobs submitted to a commercial centralized GPU cluster.

Based on the above observations, we developed **NeuronBlocks**, a DNN toolkit for NLP tasks. The basic idea is to provide two layers of support to the engineers. The upper layer targets common NLP tasks. For each task, the toolkit contains several end-to-end network templates, which can be immediately instantiated with simple configuration. The bottom layer consists of a suite of reusable and standard components, which can be adopted as building blocks to construct networks with complex architecture. By following the interface guidelines, users can also contribute to this gallery of components with their own modules.

The technical contributions of NeuronBlocks are summarized into the following three aspects.

- **Block Zoo**: categorize and abstract the most commonly used DNN components into standard and reusable blocks. The blocks within the same category can be used exchangeably.
- **Model Zoo**: identify the most popular NLP tasks and provide alternative end-to-end network templates (in JSON format) for each task.
- **Platform Compatibility**: support both Linux and Windows machines, CPU/GPU chips, as well as GPU platforms such as PAI³.

³<https://github.com/Microsoft/pai>

2 Related Work

There are several general-purpose deep learning frameworks, such as TensorFlow, PyTorch and Keras, which have gained popularity in NLP community. These frameworks offer huge flexibility in DNN model design and support various NLP tasks. However, building models under these frameworks requires a large overhead of mastering these framework details. Therefore, higher level abstraction to hide the framework details is favored by many engineers.

There are also several popular deep learning toolkits in NLP, including OpenNMT (Klein et al., 2017), AllenNLP (Gardner et al., 2018) etc. OpenNMT is an open-source toolkit mainly targeting neural machine translation or other natural language generation tasks. AllenNLP provides several pre-built models for NLP tasks, such as semantic role labeling, machine comprehension, textual entailment, etc. Although these toolkits reduce the development cost, they are limited to certain tasks, and thus not flexible enough to support new network architectures or new components.

3 Design

Neuronblocks is built on PyTorch. The overall framework is illustrated in Figure 1. It consists of two layers: the **Block Zoo** and the **Model Zoo**. In Block Zoo, the most commonly used components of neural networks are categorized into several groups according to their functions. Within each category, several alternative components are encapsulated into standard and reusable blocks with a consistent interface. These blocks serve as basic and exchangeable units to construct complex network architectures for different tasks. In Model Zoo, the most popular NLP tasks are identified. For each task, several end-to-end network templates are provided in the form of JSON configuration files. Users can simply browse these configurations and choose one to instantiate. The whole task can be completed without any coding efforts.

3.1 Block Zoo

We recognize the following major functional categories of neural network components. Each category covers as many commonly used modules as possible. The Block Zoo is an open framework, and more modules can be added in the future.

- **Embedding Layer**: Word/character embedding and extra handcrafted feature embedding

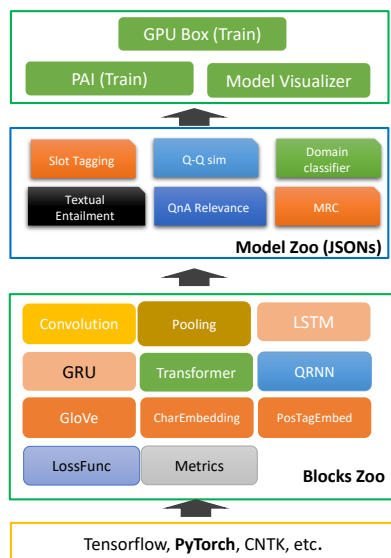


Figure 1: The overall framework of NeuronBlocks.

such as pos-tagging are supported.

- **Neural Network Layers:** Block zoo provides common layers like RNN, CNN, QRNN (Bradbury et al., 2017), Transformer (Vaswani et al., 2017), Highway network, Encoder Decoder architecture, etc. We also support multiple attention layers, such as Linear/Bi-linear Attention, Bidirectional attention flow (Seo et al., 2017), etc. Meanwhile, regularization layers such as Dropout, Layer Norm, etc are also supported.
- **Loss Function:** All built-in loss functions in PyTorch are supported.
- **Metrics:** For classification task, Area Under Curve (AUC), Accuracy, F1 metrics are supported. For sequence labeling task, F1/Accuracy are supported. For knowledge distillation task, MSE/RMSE are supported. For MRC task, ExactMatch/F1 are supported.

3.2 Model Zoo

In NeuronBlocks, we identify four types of most popular NLP tasks. For each task, we provide various end-to-end network templates.

- **Text Classification and Matching.** Tasks such as domain/intent classification, question answer matching are supported.
- **Sequence Labeling.** Predict each token in a sequence into predefined types. Common tasks include NER, POS tagging, Slot tagging, etc.
- **Knowledge Distillation (Hinton et al., 2015).** Teacher-Student based knowledge distillation is one common approach for model compression. NeuronBlocks provides knowledge

distillation template to train light-weight student model to imitate heavy DNN models like BERT/GPT.

- **Extractive Machine Reading Comprehension.** Given question and passage, this task is to predict the start and end positions of the answer spans in the passage.

3.3 User Interface

NeuronBlocks provides convenient user interface⁴ for users to build, train, and test DNN models. The details are described in the following.

- **I/O interface.** This part defines model input/output, such as training data, pre-trained models/embeddings, model saving path, etc.
- **Model Architecture interface.** This is the key part of the configuration file, which defines the whole model architecture. Figure 2 shows an example of how to specify a model architecture using the blocks in NeuronBlocks. To be more specific, it consists of a list of layers/blocks to construct the architecture, where the blocks are supplied in the gallery of *Block Zoo*.
- **Training Parameters interface.** In this part, the model optimizer as well as all other training hyper parameters are indicated.

3.4 Workflow

Figure 3 shows the workflow of building DNN models in NeuronBlocks. Users only need to write a JSON configuration file. They can either instantiate an existing template from *Model Zoo*, or construct a new architecture based on the blocks from *Block Zoo*. This configuration file is shared across training, test, and prediction.

For model hyper-parameter tuning or architecture modification, users just need to change the JSON configuration file. Advanced users can also contribute novel customized blocks⁵ into *Block Zoo*, as long as they follow the same interface guidelines with the existing blocks. These new blocks can be further shared across all users for model architecture design. Moreover, NeuronBlocks has flexible platform support, such as GPU/CPU, GPU management platforms like PAI.

4 Experiments

To verify the performance of NeuronBlocks, we conducted extensive experiments for common

⁴<https://github.com/microsoft/NeuronBlocks/blob/master/Tutorial.md>

⁵<https://github.com/microsoft/NeuronBlocks/blob/master/Contributing.md>

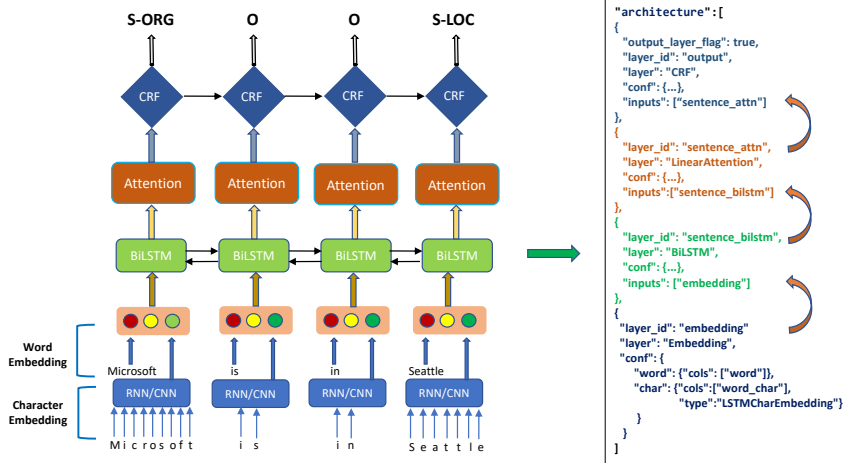


Figure 2: A Model architecture interface example of sequence labeling model in NeuronBlocks.

Results(F1-score)		WLSTM+CRF	WLSTM	WCNN+CRF	WCNN
Nochar	Literature	89.31 \pm 0.1(N)	87.00(M-16) 88.49 \pm 0.17(N)	88.65 \pm 0.2(N)	88.50 \pm 0.05(N)
	NeuronBlocks	89.34	88.50	88.72	88.51
CLSTM	Literature	90.94(L-16) 91.08 \pm 0.08(N)	89.15(L-16) 90.77 \pm 0.06(N)	90.48 \pm 0.23(N)	90.28 \pm 0.30(N)
	NeuronBlocks	91.03	90.67	90.27	90.37
CCNN	Literature	90.91 \pm 0.2(C-16) 91.21(M-16) 90.87 \pm 0.13(P-17) 91.11 \pm 0.21(N)	89.36(M-16) 90.60 \pm 0.11(N)	90.28 \pm 0.09(N)	90.51 \pm 0.19(N)
	NeuronBlocks	91.38	90.63	90.41	90.36

Table 2: NeuronBlocks results on CoNLL-2003 English NER testb dataset. The abbreviation (C-16)= (Chiu and Nichols, 2016), (L-16)= (Lample et al., 2016), (M-16)= (Ma and Hovy, 2016), (N)= (Yang et al., 2018), (P-17)= (Peters et al., 2017).

Model	CoLA	SST-2	QQP	MNLI	QNLI	RTE	WNLI
BiLSTM (Literature)	17.6	87.5	85.3/82.0	66.7	77.0	58.5	56.3
+Attn (Literature)	17.6	87.5	87.7/83.9	70.0	77.2	58.5	60.6
BiLSTM (NeuronBlocks)	20.4	87.5	86.4/83.1	69.8	79.8	59.2	59.2
+Attn (NeuronBlocks)	25.1	88.3	87.8/83.9	73.6	81.0	58.9	59.8

Table 3: NeuronBlocks results on GLUE benchmark development sets. As described in (Wang et al., 2019), for CoLA, we report Matthews correlation. For QQP, we report accuracy and F1. For MNLI, we report accuracy averaged over the matched and mismatched development sets. For all other tasks we report accuracy. All values have been scaled by 100. Please note that results on the development sets are reported, since GLUE does not distribute labels for the test sets.

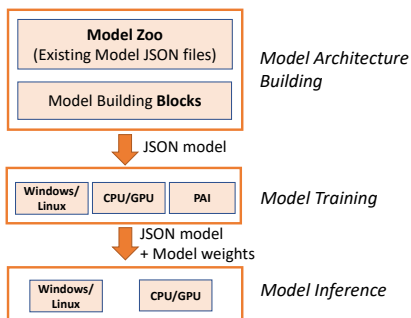


Figure 3: The workflow of NeuronBlocks.

NLP tasks on public data sets including CoNLL-2003 (Sang and Meulder, 2003), GLUE benchmark (Wang et al., 2019), and WikiQA corpus (Yang et al., 2015). The experimental results showed that the models built with NeuronBlocks can achieve reliable and competitive results on

various tasks, with productivity greatly improved.

4.1 Sequence Labeling

We evaluated NeuronBlocks on CoNLL-2003 (Sang and Meulder, 2003) English NER dataset, following most works on the same task. This dataset includes four types of named entities, namely, *PERSON*, *LOCATION*, *ORGANIZATION*, and *MISC*. We adopted BIOES tagging scheme instead of IOB, as many previous works indicated meaningful improvement with BIOES scheme (Ratinov and Roth, 2009). Table 2 shows the results on CoNLL-2003 English testb dataset, with 12 different combinations of network layers/blocks, such as word/character embedding, CNN/LSTM and CRF. The results suggest that the flexible combination of layers/blocks in Neu-

Model	Inference Speed QPS	Parameters	Performance AUC
Teacher Model (BERT _{base})	448	110M	0.9112
Student Model (BiLSTMAttn+TextCNN)	11128	13.63M	0.8941

Table 4: NeuronBlocks results on Knowledge Distillation task.

ronBlocks can easily reproduce the performance of original models, with comparative or slightly better performance.

4.2 GLUE Benchmark

The General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2019) is a collection of natural language understanding tasks. We experimented on the GLUE benchmark tasks using BiLSTM and Attention based models. As shown in Table 3, the models built by NeuronBlocks can achieve competitive or even better results on GLUE tasks with minimal coding efforts.

4.3 Knowledge Distillation

In teacher-student based knowledge distillation, a lightweight student model is trained by imitating the output of heavy teacher model like BERT. We evaluated on a binary classification dataset called Domain Classification Dataset, which was collected from a commercial search engine. Each sample in this dataset consists of two parts, i.e., a query and a binary label indicating whether the query belongs to a specific domain. Table 4 shows the results, where AUC is used as the evaluation criteria and Queries per Second (QPS) is used to measure inference speed. By leveraging knowledge distillation, the student model by NeuronBlocks managed to get **24.8** times inference speedup with only small performance regression compared with BERT_{base}⁶ fine-tuned classifier.

4.4 WikiQA

The WikiQA corpus (Yang et al., 2015) is a publicly available dataset for open-domain question answering. This dataset contains 3,047 questions from Bing query logs, each associated with some candidate answer sentences from Wikipedia. We conducted experiments on WikiQA dataset using CNN, BiLSTM, and Attention based models. The results are shown in Table 5. The models built in NeuronBlocks achieved competitive or even better results with simple model configurations.

5 Conclusion and Future Work

In this paper, we introduce NeuronBlocks, a DNN toolkit for NLP tasks built on PyTorch, targeting

⁶<https://github.com/huggingface/pytorch-transformers>

Model	AUC
CNN ((Yang et al., 2015))	73.59
CNN-Cnt ((Yang et al., 2015))	75.33
CNN (NeuronBlocks)	74.79
BiLSTM (NeuronBlocks)	76.73
BiLSTM+Attn (NeuronBlocks)	75.48
BiLSTM+MatchAttn (NeuronBlocks)	78.54

Table 5: NeuronBlocks results on WikiQA.

three types of engineers, and provides a two-layer solution to satisfy the requirements from all three types of users. To be more specific, the *Model Zoo* consists of various templates for the most common NLP tasks, while the *Block Zoo* supplies a gallery of alternative layers/modules for the networks. Such design achieves a balance between generality and flexibility. Extensive experiments have verified the effectiveness of this approach. NeuronBlocks has been widely used in a product team of a commercial search engine, and significantly improved the productivity for developing NLP DNN approaches.

As an open-source toolkit, we will further extend it in various directions. The following names a few examples.

- Multi-task learning (MTL). In MTL, multiple related tasks are jointly trained so that knowledge learned in one task can benefit other tasks.
- Pre-training and fine-tuning. Deep pre-training models such as ELMo (Peters et al., 2018), GPT (Radford et al., 2018), BERT (Devlin et al., 2019) are new directions in NLP.
- Sequence generation task. Sequence generation is widely used in NLP fields such as machine translation (Bahdanau et al., 2015), text summarization (See et al., 2017), and dialogue systems (Wen et al., 2015).
- AutoML (Elsken et al., 2019). NeuronBlocks facilitates users to build models on top of *Model Zoo* and *Block Zoo*. With the integration of AutoML, the toolkit can further support automatic model architecture design.

6 Acknowledgements

We sincerely thank the anonymous reviewers for their valuable suggestions.

References

- Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate.
- James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. 2017. Quasi-recurrent neural networks.
- Jason P. C. Chiu and Eric Nichols. 2016. Named entity recognition with bidirectional lstm-cnns. *TACL*, 4:357–370.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. pages 4171–4186.
- Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21.
- Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. Allennlp: A deep semantic natural language processing platform. *arXiv preprint arXiv:1803.07640*.
- Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531.
- Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. 2017. Opennmt: Open-source toolkit for neural machine translation. pages 67–72.
- Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 260–270.
- Xuezhe Ma and Eduard H. Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*.
- Matthew E. Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. 2017. Semi-supervised sequence tagging with bidirectional language models. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1756–1765.
- Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. pages 2227–2237.
- Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. URL https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/languageunsupervised/language_understanding_paper.pdf.
- Lev-Arie Ratinov and Dan Roth. 2009. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning, CoNLL 2009, Boulder, Colorado, USA, June 4-5, 2009*, pages 147–155.
- Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning, CoNLL 2003, Held in cooperation with HLT-NAACL 2003, Edmonton, Canada, May 31 - June 1, 2003*, pages 142–147.
- Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. pages 1073–1083.
- Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2017. Bidirectional attention flow for machine comprehension.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.
- Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.
- Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Peihao Su, David Vandyke, and Steve J. Young. 2015. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. pages 1711–1721.
- Jie Yang, Shuailong Liang, and Yue Zhang. 2018. Design challenges and misconceptions in neural sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018*, pages 3879–3889.
- Yi Yang, Wen-tau Yih, and Christopher Meek. 2015. Wikiqa: A challenge dataset for open-domain question answering. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2013–2018.