NeuronBlocks – Building Your NLP DNN Models Like Playing Lego

Ming Gong[§], Linjun Shou[§], Wutao Lin[§], Zhijie Sang[Ł], Quanjia Yan[ł], Ze Yang[Ł], Daxin Jiang[§]

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

[Ł]Center for Intelligence Science and Technology, BUPT, Beijing, China

[ł]Research Center for Ubiquitous Computing Systems, Institute of Computing Technology, CAS, Beijing, China

{migon, lisho, wutlin, djiang}@microsoft.com
{woailaosang, yangze01}@bupt.edu.cn
yanquanjia17s@ict.ac.cn

Abstract

When building deep neural network models for natural language processing tasks, engineers often spend a lot of efforts on coding details and debugging, instead of focusing on model architecture design and hyper-parameter tuning. In this paper, we introduce NeuronBlocks, a deep neural network toolkit for natural language processing tasks. In NeuronBlocks, a suite of neural network layers are encapsulated as building blocks, which can easily be used to build complicated deep neural network models by configuring a simple JSON file. NeuronBlocks empowers engineers to build and train various NLP models in seconds even without a single line of code. A series of experiments on real NLP datasets such as GLUE and WikiQA have been conducted, which demonstrates the effectiveness of NeuronBlocks.

1 Introduction

Deep neural network (DNN) models have been widely used for solving various natural language processing (NLP) tasks, such as text classification, slot tagging, question answering, etc. However, when engineers try to address specific NLP tasks with DNN models, they often face the following challenges:

1) Lots of DNN frameworks to choose and high studying cost, such as TensorFlow, PyTorch, Keras, etc.
2) Heavy coding cost. Too many details make DNN model code hard to debug.
3) Fast Model Architecture Evolution. It is difficult for engineers to understand the mathematical principles behind them.
4) Model Code optimization requires deep expertise.
5) Model Platform Compatibility Requirement. It requires extra coding work for the model to run on different platforms, such as Linux/Windows, GPU/CPU.

With the above challenges, engineers often spend too much time on code writing and debugging, which limits the efficiency of model building. Our practice shows that in real applications, it is usually more important for engineers to focus on data preparation rather than model implementation, since high quality data often brings more improvement than small variation of model itself. Moreover, simple and standard models are often more robust than complicated variations of models and are also easy to maintain, share, and reuse. For those tasks requiring complex models, the agility of model iteration becomes the key. Once training data has been well collected, it is critical to try different architectures and parameters with small efforts.

Through analyzing NLP jobs submitted to a commercial centralized GPU cluster, we found that about 87.5% NLP related jobs belong to common tasks, such as sentence classification, sequence labelling etc. By further analysis, we found that more than 90% of models can be composed of common components like embedding layer, CNN/RNN, Transformer etc. Based on this finding, our thinking is that whether it is possible to provide a suite of reusable and standard components, as well as a set of popular models for common tasks. This would greatly boost model training productivity by saving lots of duplicate efforts.

Based on the above motivations, we developed NeuronBlocks[1], a DNN toolkit for NLP tasks,

[1] https://github.com/Microsoft/NeuronBlocks

which provides a suite of reusable and standard components, as well as a set of popular models for common tasks. NeuronBlocks minimizes duplicated efforts of code writing during DNN model training. With NeuronBlocks, users only need to define a simple JSON configuration file to design and train models, which enables engineers to focus more on high level model design, instead of programming language and framework details.

The technical contributions of NeuronBlocks are summarized as three aspects:

- ***Block Zoo***: abstract the most commonly used components of DNN into standard and reusable blocks.
- ***Model Zoo***: provide a rich scope of model architectures covering popular NLP tasks.
- ***Platform Compatibility***: support both Linux and Windows machines, CPU/GPU chips, as well as GPU platforms like PAI[2].

2 Related Work

Currently, there are several general-purpose deep learning frameworks, such as TensorFlow (Abadi et al., 2016), PyTorch (Paszke et al., 2017) and Keras (Chollet, 2015), which have gained popularity in NLP community. These frameworks offer a huge flexibility in DNN model design and support various NLP tasks. However, building model with these frameworks still require heavy cost to learn framework details. Therefore, we need higher levels of abstraction for the purpose of skipping the framework details. Other Popular deep learning toolkits in NLP include OpenNMT (Klein, Kim, Deng, Senellart, & Rush, 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 like semantic role labeling, machine comprehension, textual entailment, etc. However, to create a new model usually requires complex programming work.

3 Design

NeuronBlocks is an NLP deep learning modeling toolkit which help engineers to easily build end-to-end pipelines for neural network model training. The main goal is to minimize developing cost for DNN model building all stages, including training, test and inference.

NeuronBlocks is built on PyTorch[3], including two major components: *Block Zoo* and *Model Zoo*. In *Block Zoo*, we abstract and encapsulate the most commonly used components of deep neural networks into standard and reusable blocks. These block units act like building blocks to form various complicated neural network architectures for different NLP tasks. In *Model Zoo*, we select a rich scope of model architectures covering popular NLP tasks in the form of simple JSON configuration files. With *Model Zoo*, users can easily navigate these model configurations and select propriate models for their target task, modify a few configurations if needed, and then start training immediately without any coding efforts.

3.1 Overall Framework

The overall framework of NeuronBlocks is shown in Figure 1 which consists of two major components: *Block Zoo* and *Model Zoo*. In *Block Zoo*, we provide commonly used neural network components as building blocks for model architecture design. *Model Zoo* consists of various DNN models for common NLP tasks, in the form of JSON configuration files.

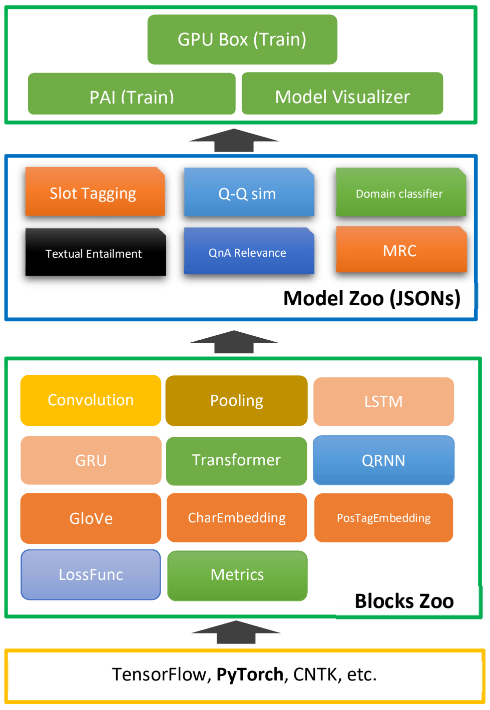

Figure 1. NeuronBlocks Overall Framework

[2] https://github.com/Microsoft/pai
[3] https://pytorch.org

3.2 Block Zoo

Blocks for deep learning models are like bricks for a tall building. Block Zoo consists of key components of neural networks. In current released version, blocks supported by NeuronBlocks are summarized as follows (more will be added):

- Embedding Layer: Word embedding, character embedding and POS tagging embedding are supported.
- Neural Network Layers: LSTM (Hochreiter & Schmidhuber, 1997), GRU (Chung, Gulcehre, Cho, & Bengio, 2014), Bidirectional LSTM, Bidirectional GRU, QRNN (Bradbury, Merity, Xiong, & Socher, 2016), Bidirectional QRNN, Transformer (Vaswani et al., 2017), CNN, Pooling, Highway network (R. K. Srivastava, Greff, & Schmidhuber, 2015), Encoder Decoder architecture, etc.
- Attention Mechanism: Linear Attention, Bilinear Attention, Full Attention (Huang, Zhu, Shen, & Chen, 2017), Bidirectional attention flow (Seo, Kembhavi, Farhadi, & Hajishirzi, 2016), etc.
- Mathematical Operations: Add, Minus, Elementwise Multiply, Matrix Multiply, etc.
- Regularization Layers: Dropout (N. Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014), Layer Norm, etc.
- Optimizers: All the optimizers defined in PyTorch are supported by default.
- Loss Function: All the loss functions offered by PyTorch are supported. Additionally, we offer more options, such as Focal Loss (Lin, Goyal, Girshick, He, & Dollár, 2017).
- Metrics: For classification task, AUC, Accuracy, Recall, Precision, F1 metrics are supported. For sequence tagging task, F1 and Accuracy are supported. For regression task, MSE and RMSE are supported. For machine reading comprehension task, Exact Match and F1 are supported.

3.3 Model Zoo

NeuronBlocks supports 4 major NLP tasks (more will be added):

- Text classification, including single sentence, sentence pair classification, such as query domain classification, intent classification, natural language inference, etc.
- Sequence tagging. Given a sentence, classify each token in the sequence into some predefined categories. Common tasks include NER, POS tagging, slot tagging, etc.
- Regression task. Unlike classification, the output of a regression task is a continuous real number.
- Extractive machine reading comprehension. Given a question and a passage, predict whether a span from the passage a direct answer to the question or not.

3.4 Model Configuration

To train a DNN model or inference with one exiting model, users need to define model architecture, training data and hyper-parameters in a JSON configuration file. In this subsection, we will introduce the details of the configuration file through an example for question answer matching task. The configuration file consists of Inputs, Outputs, Training Parameters, Model Architecture, Loss, Metrics, etc.

- **Inputs**: This part defines the data input configuration, including training data location, data schema, model inputs, model targets, etc.

```
"inputs": {
  "use_cache": true,
  "dataset_type": "classification",
  "data_paths": {
    "train_data_path":
"./dataset/demo/train.tsv",
    "valid_data_path":
"./dataset/demo/valid.tsv",
    "test_data_path":
"./dataset/demo/test.tsv",
    "predict_data_path":
"./dataset/demo/predict.tsv",
    "pre_trained_emb":
"./dataset/GloVe/glove.840B.300d.txt"
  },
  "file_with_col_header": false,
  "pretrained_emb_type": "glove",
  "pretrained_emb_binary_or_text": "text",
  "involve_all_words_in_pretrained_emb":
false,
  "add_start_end_for_seq": true,
  "file_header": {
    "question_text": 0,
    "answer_text": 1,
    "label": 2
  },
  "predict_file_header": {
    "question_text": 0,
    "answer_text": 1
  },
  "model_inputs": {
    "question": ["question_text"],
    "answer": ["answer_text"]
  },
  "target": ["label"]
}
```

- **Outputs:** This part defines model saving paths, cache paths, log paths, etc.

```
"outputs": {
  "save_base_dir": "./models/demo/",
  "model_name": "model.nb",
  "train_log_name": "train.log",
  "test_log_name": "test.log",
  "predict_log_name": "predict.log",
  "predict_fields": ["prediction",
"confidence"],
  "predict_output_name": "predict.tsv",
  "cache_dir": ".cache.demo/"
}
```

- **Training Parameters:** In this part, model optimizer and all other training hyper parameters are indicated. All optimizers defined in PyTorch are supported.

```
"training_params": {
  "optimizer": {
    "name": "Adam",
    "params": {
      "lr": 0.001
    }
  },
  "use_gpu": false,
  "batch_size": 30,
  "batch_num_to_show_results": 10,
  "max_epoch": 3,
  "valid_times_per_epoch": 1,
  "max_lengths": {
    "question_text": 30,
    "answer_text": 100
  }
}
```

- **Model Architecture:** This part is the key component of the configuration file, which define the whole model architecture. Basically, it consists of a list of layers/blocks to form the architecture, based on the blocks supported in the *Block Zoo*.

```
"architecture": [
  {
    "layer": "Embedding",
    "conf": {
      "word": {
        "cols": ["question_text",
"answer_text"],
        "dim": 300
      }
    }
  },
  {
    "layer_id": "question_1",
    "layer": "BiLSTM",
    "conf": {
      "hidden_dim": 64,
      "dropout": 0.2,
      "num_layers": 2
    },
    "inputs": ["question"]
  },
  {
    "layer_id": "answer_1",
    "layer": "BiLSTM",
    "conf": {
      "hidden_dim": 64,
      "dropout": 0.2,
      "num_layers": 2
    },
    "inputs": ["answer"]
  },
  {
    "layer_id": "question_2",
    "layer": "Pooling",
    "conf": {
      "pool_axis": 1,
      "pool_type": "max"
    },
    "inputs": ["question_1"]
  },
  {
    "layer_id": "answer_2",
    "layer": "question_2",
    "inputs": ["answer_1"]
  },
  {
    "layer_id": "comb",
    "layer": "Combination",
    "conf": {
      "operations": ["origin",
"difference", "dot_multiply"]
    },
    "inputs": ["question_2", "answer_2"]
  },
  {
    "output_layer_flag": true,
    "layer_id": "output",
    "layer": "Linear",
    "conf": {
      "hidden_dim": [128, 2],
      "activation": "PReLU",
      "batch_norm": true,
      "last_hidden_activation": false,
      "last_hidden_softmax": false
    },
    "inputs": ["comb"]
  }
]
```

- **Loss:** Loss function of the model is defined here.

```
"loss": {
  "losses": [
    {
      "type": "CrossEntropyLoss",
      "conf": {
        "size_average": true
      },
      "inputs": ["output", "label"]
    }
  ]
}
```

- **Metrics:** Task metrics are defined here. Users can define a metric list that they want to check during model testing and validation stages. The first metric in the metric list is what our toolkit uses to select the best model during training.

```
"metrics": ["auc","accuracy"]
```

3.5 Workflow

The whole workflow of leveraging NeuronBlocks for model building is quite simple. Users only need to write a JSON configuration file, by either using existing models from *Model Zoo* or building new models with blocks from *Block Zoo*. This configuration file is shared for training, testing, and prediction. The commands are as follows:

```
python train.py   -conf_path=conf.json
python test.py    -conf_path=conf.json
python predict.py -conf_path=conf.json
```

For model hyper-parameter tuning or model structure modification, just change the JSON config file. Experienced users can even add new customized blocks to *Block Zoo*. Then these new blocks can be easily used for model architecture design. NeuronBlocks also supports model training on GPU management platform like PAI.

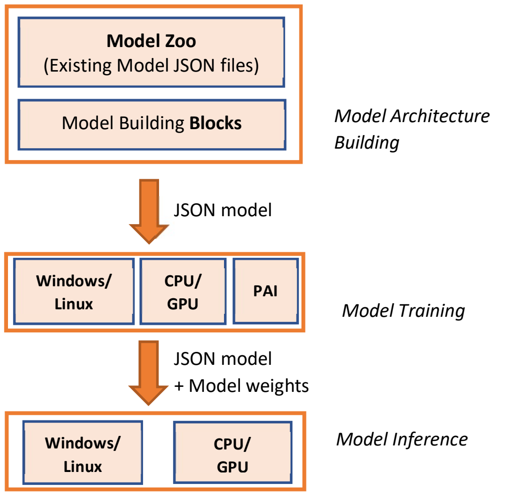

Figure 2. NeuronBlocks Workflow

3.6 Model Visualizer

To check the correctness of model configuration and visualize model architecture, NeuronBlocks provides a visualization tool. For example, the model graph of the configuration file we mentioned in Section 3.4 is visualized as Figure 3.

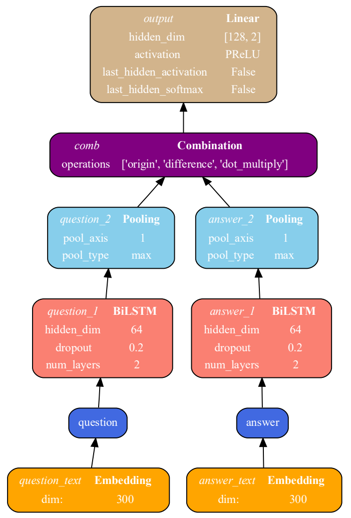

Figure 3. Model Visualizer Result of a sample model.

4 Experiments

To verify the performance of NeuronBlocks, we conducted several experiments on some common NLP tasks like GLUE benchmark (Wang et al., 2018) and WikiQA corpus (Yang, Yih, & Meek, 2015). Results show that models built with NeuronBlocks can get reliable and competitive results on various tasks, with productivity greatly boosted.

4.1 GLUE Benchmark

The General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018) is a collection of various natural language understanding tasks. We performed experiments on the GLUE benchmark tasks using BiLSTM and Attention based models. We report NeuronBlocks' results on the development sets, since GLUE does not distribute labels for the test sets. The detailed results are showed in Table 1.

Model	CoLA	SST-2	QQP	MNLI	QNLI	RTE	WNLI
BiLSTM (paper)	17.6	87.5	85.3/82.0	66.7	77.0	58.5	56.3
+Attn (paper)	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 1: NeuronBlocks' results on the GLUE benchmark' development sets. As described in (Wang et al., 2018), 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.

4.2 WikiQA

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. As shown in Table 2, we conducted experiments on WikiQA dataset, using CNN, BiLSTM, and Attention based models. Area Under Curve (AUC) metric is used as the evaluation criteria.

Model	AUC
CNN (paper baseline)	73.59
CNN-Cnt (paper baseline)	75.33
CNN (NeuronBlocks)	**74.79**
BiLSTM (NeuronBlocks)	**76.73**
BiLSTM+Attn (NeuronBlocks)	**75.48**

Table 2. NeuronBlocks' results on WikiQA.

5 Future Work

NeuronBlocks is an open-source toolkit, with the following work to be done but not limited to:

- Multi-Task training support. Currently NeuronBlocks only supports single task model training. We plan to add support for multi-task training soon.
- Pretraining and Finetuning support. Deep pretraining models such as ELMo (Peters et al., 2018), OpenAI GPT (Radford, Narasimhan, Salimans, & Sutskever, 2018), BERT (Devlin, Chang, Lee, & Toutanova, 2018) are new directions in NLP. We will add support for these models as well.
- Multi-Lingual support. NeuronBlocks only supports English language for now. In the future, other languages will be supported as well.
- Knowledge distillation for heavy models such as BERT, OpenAI Transformer. Deep pretraining models have shown excellent results on plenty of NLP tasks. However, inference speed is usually very slow for production usage. Teacher-Student based knowledge distillation is one common method for model compression.
- AutoML support for automatic neural network architecture search. Currently, NeuronBlocks provides users with easy-to-use experience to build models on top of *Model Zoo* and *Block Zoo*. With AutoML support, automatic model architecture design can be achieved for specific task and data.

6 Conclusion

In this paper, we introduce NeuronBlocks, a DNN toolkit for NLP tasks built on PyTorch. NeuronBlocks provides easy-to-use model training and inference experience for users. Model building time can be significantly reduced from days to hours, even minutes. We also provide a suite of pre-built DNN models for popular NLP tasks in *Model Zoo*. Majority of the users can simply pick one from *Model Zoo* to start model training. For some experienced users, they can also create new blocks to *Block Zoo*, which will greatly benefit the top users for model building. Besides, a series of experiments on real NLP tasks have been conducted to verify the effectiveness of this toolkit.

References

Bradbury, J., Merity, S., Xiong, C., & Socher, R. (2016). Quasi-recurrent neural networks. *ArXiv Preprint ArXiv:1611.01576*.

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *ArXiv Preprint ArXiv:1412.3555*.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv Preprint ArXiv:1810.04805*.

Gardner, M., Grus, J., Neumann, M., Tafjord, O., Dasigi, P., Liu, N., … Zettlemoyer, L. (2018). AllenNLP: A deep semantic natural language processing platform. *ArXiv Preprint ArXiv:1803.07640*.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Huang, H.-Y., Zhu, C., Shen, Y., & Chen, W. (2017). Fusionnet: Fusing via fully-aware attention with application to machine comprehension. *ArXiv Preprint ArXiv:1711.07341*.

Klein, G., Kim, Y., Deng, Y., Senellart, J., & Rush, A. M. (2017). Opennmt: Open-source toolkit for neural machine translation. *ArXiv Preprint ArXiv:1701.02810*.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision* (pp. 2980–2988).

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. *ArXiv Preprint ArXiv:1802.05365*.

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (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.*

Seo, M., Kembhavi, A., Farhadi, A., & Hajishirzi, H. (2016). Bidirectional attention flow for machine comprehension. *ArXiv Preprint ArXiv:1611.01603*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, *15*(1), 1929–1958.

Srivastava, R. K., Greff, K., & Schmidhuber, J. (2015). Highway networks. *ArXiv Preprint ArXiv:1505.00387*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems* (pp. 5998–6008).

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. R. (2018). Glue: A multi-task benchmark and analysis platform for natural language understanding. *ArXiv Preprint ArXiv:1804.07461*.

Yang, Y., Yih, W., & Meek, C. (2015). Wikiqa: A challenge dataset for open-domain question answering. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 2013–2018).

Appendix

Here is a complete example of JSON config file:

```
{
  "license": "Copyright (c) Microsoft Corporation.
All rights reserved. Licensed under the MIT
license.",
  "tool_version": "1.1.0",
  "model_description": "This example shows how to
train/test/predict.",
  "inputs": {
    "use_cache": true,
    "dataset_type": "classification",
    "data_paths": {
      "train_data_path":
"./dataset/demo/train.tsv",
      "valid_data_path":
"./dataset/demo/valid.tsv",
      "test_data_path": "./dataset/demo/test.tsv",
      "predict_data_path":
"./dataset/demo/predict.tsv",
      "pre_trained_emb":
"./dataset/GloVe/glove.840B.300d.txt"
    },
    "file_with_col_header": false,
    "pretrained_emb_type": "glove",
    "pretrained_emb_binary_or_text": "text",
    "involve_all_words_in_pretrained_emb": false,
    "add_start_end_for_seq": true,
    "file_header": {
      "question_text": 0,
      "answer_text": 1,
      "label": 2
    },
    "predict_file_header": {
      "question_text": 0,
      "answer_text": 1
    },
    "model_inputs": {
      "question": ["question_text"],
      "answer": ["answer_text"]
    },
    "target": ["label"]
  },
  "outputs": {
    "save_base_dir": "./models/demo/",
    "model_name": "model.nb",
    "train_log_name": "train.log",
    "test_log_name": "test.log",
    "predict_log_name": "predict.log",
    "predict_fields": ["prediction",
"confidence"],
    "predict_output_name": "predict.tsv",
    "cache_dir": ".cache.demo/"
  },
  "training_params": {
    "optimizer": {
      "name": "Adam",
      "params": {
        "lr": 0.001
      }
    },
    "use_gpu": false,
    "batch_size": 30,
    "batch_num_to_show_results": 10,
    "max_epoch": 3,
    "valid_times_per_epoch": 1,
    "max_lengths": {
      "question_text": 30,
      "answer_text": 100
    }
  },
  "architecture": [
    {
      "layer": "Embedding",
      "conf": {
        "word": {
          "cols": ["question_text",
"answer_text"],
          "dim": 300
        }
      }
    },
    {
      "layer_id": "question_1",
      "layer": "BiLSTM",
      "conf": {
        "hidden_dim": 64,
        "dropout": 0.2,
        "num_layers": 2
      },
      "inputs": ["question"]
    },
    {
      "layer_id": "answer_1",
      "layer": "BiLSTM",
      "conf": {
        "hidden_dim": 64,
        "dropout": 0.2,
        "num_layers": 2
      },
      "inputs": ["answer"]
    },
    {
      "layer_id": "question_2",
      "layer": "Pooling",
      "conf": {
        "pool_axis": 1,
        "pool_type": "max"
      },
      "inputs": ["question_1"]
    },
    {
      "layer_id": "answer_2",
      "layer": "question_2",
      "inputs": ["answer_1"]
    },
    {
      "layer_id": "comb",
      "layer": "Combination",
      "conf": {
        "operations": ["origin", "difference",
"dot_multiply"]
      },
      "inputs": ["question_2", "answer_2"]
    },
    {
      "output_layer_flag": true,
      "layer_id": "output",
      "layer": "Linear",
      "conf": {
        "hidden_dim": [128, 2],
        "activation": "PReLU",
        "batch_norm": true,
        "last_hidden_activation": false,
        "last_hidden_softmax": false
      },
      "inputs": ["comb"]
    }
  ],
  "loss": {
    "losses": [
      {
        "type": "CrossEntropyLoss",
        "conf": {
          "size_average": true
        },
        "inputs": ["output", "label"]
      }
    ]
  },
  "metrics": ["auc", "accuracy"]
}
```