DELTA
A DEep learning based Language Technology plAtform

Kun Han, Junwen Chen, Hui Zhang, Haiyang Xu, Yiping Peng, Yun Wang, Ning Ding, Hui Deng, Yonghu Gao, Tingwei Guo, Yi Zhang, Yahao He, Baochang Ma, Yulong Zhou, Kangli Zhang, Chao Liu, Ying Lyu, Chenxi Wang, Cheng Gong, Yunbo Wang, Wei Zou, Hui Song, and Xiangang Li

AI Labs, DiDi Chuxing[*]

Abstract

In this paper we present DELTA, a deep learning based language technology platform. DELTA is an end-to-end platform designed to solve industry level natural language and speech processing problems. It integrates most popular neural network models for training as well as comprehensive deployment tools for production. DELTA aims to provide easy and fast experiences for using, deploying, and developing natural language processing and speech models for both academia and industry use cases. We demonstrate the reliable performance with DELTA on several natural language processing and speech tasks, including text classification, named entity recognition, natural language inference, speech recognition, speaker verification, etc. DELTA has been used for developing several state-of-the-art algorithms for publications and delivering real production to serve millions of users.

1 Introduction

In recent years, deep learning has been achieving tremendous success on numerous machine learning applications (LeCun et al., 2015; Schmidhuber, 2015). It firstly dramatically improved the state-of-the-art in speech recognition (Hinton et al., 2012) and image recognition (Krizhevsky et al., 2012), and then produced extremely promising results on natural language processing (NLP) (Collobert et al., 2011; Devlin et al., 2019; Sutskever et al., 2014).

In natural language and speech processing areas, various algorithms and models are emerging recently due to the rapid progress in both academia and industry. Recently, many open-sourced libraries are released for community development since the interest in applying deep learning approaches to NLP and speech processing is very high. However, some open-sourced libraries are implemented for specific tasks, and researchers and engineers have to adapt the code for their own problems. More importantly, most existing implementations are designed for research purposes and do not consider model serving. Although TensorFlow provides interfaces for model serving, it is not trivial to deploy the models for complex production usage.

We introduce DELTA,[2] a deep learning based natural language and speech processing platform, as an end-to-end open-sourced library, which integrates most popular NLP and speech processing models and deployment tools, forming with uniform code structures and input/output (I/O) interfaces. DELTA is implemented using TensorFlow (Abadi et al., 2015), featuring easy-to-use, easy-to-deploy, and easy-to-develop:

- Easy to use

---

[*]Correspondence to: Kun Han, Junwen Chen, and Hui Zhang: {kunhan,chenjunwen, ethanzhanghui}@didiglobal.com
[2]https://github.com/didi/delta

- It supports most common NLP and speech tasks that works out of the box. The users may directly complete model training through one line of command.
- It is of highly-customized configuration. The users can easily customize model architectures and parameters.
- It supports multimodal training using textual, acoustic, and numeric features, which is very useful for the multimodal scenarios especially in real applications.
- It optimizes the front-end data preprocessing and parallelizes model training so it is easy to train models on a huge amount of data.

- Easy to deploy
  - What you see in training is what you get in serving. All data processing and feature extraction components are implemented as TensorFlow operators (OPs) and integrated into a graph for deployment.
  - All models have the uniform I/O interfaces and transparent to outside. One can deploy a new model with no changes on the serving code.

- Easy to develop
  - The models and components are modularized. The developers can easily build new models on top of them.
  - All modules are fully-tested and provide reliable and efficient performance.

2 Platform architecture

This section describes the overall architecture and code structures for DELTA. In high-level, DELTA includes two parts:

- Modeling: This is the main package for offline modeling, including data processing, model building, training, and evaluation, etc. The code is organized in `delta` directory. We discuss the details for this part in this section.

- Deployment: This is the package for online serving using existing model. DELTA provides flexible online serving approaches for various production environments such as GPU machines, mobile devices, embedded devices. The code is organized in `deltann` directory. We will discuss this part in the next section.
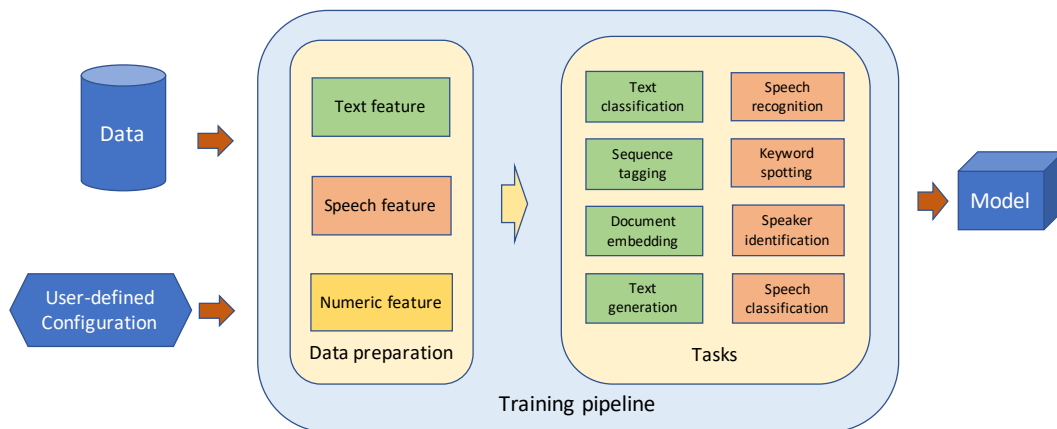

Figure 1: High-level overview of DELTA.

Fig. 1 shows a high-level overview for the modeling part in DELTA. An user is responsible for providing data and a user-defined configuration. Based on the configuration, DELTA can automatically generate a training pipeline including the corresponding data processing and modeling modules. The pipeline will start model training processing and generate a model once complete.

2.1 Data processing

Acoustic and textual information are the two most important ways for human communication, both of which are presented as sequences but in different modalities. Therefore most sequential models are applicable to both text and speech. In DELTA, we use `Dataset` API to load and process data, and the code is modularized in the directory `delta/data`[3].

2.1.1 Text data

DELTA supports text inputs in the raw text format. Preprocessing transforms the text inputs from the original format to a model-readable format. For English, we provide standard text processing functions using TensorFlow operations, including splitting, punctuation conversion, casing, etc. For Chinese, word segmentation is usually used to organize Chinese characters into words. We implement several Chinese word segmentation algorithms as TensorFlow OPs, such as *jieba*[4]. A *tokenizer* is then applied to transform a text string into a list of indices. In DELTA, `SentenceToIdsOp` is applied to index the tokens based on a vocabulary look-up table. All these processing operations are implemented as customized OPs, and some are introduced from the third-party libraries such as *Lingvo*[5] (Shen et al., 2019). The OPs are organized in `delta/layers/ops/py_x_ops.py` and can be compiled as a dynamic library.

2.1.2 Speech data

Speech tasks, unlike NLP tasks, usually come with complex feature extraction on raw speech signals. TensorFlow has provided several functions for speech processing, such as MFCC and spectrogram. In DELTA, we also implement speech processing operations for broader applications, including filter-bank feature, perceptual linear prediction feature (PLP), zero crossing rate, frame power, pitch, first order and seconder order derivatives between frames (i.e., deltas and accelerations). To perform high-level transformation, we implemented `AnalysisFilterBank` and `SynthesisFilterBank` to generate spectrogram and phase features from waveforms and reconstruct waveforms from both, respectively. We reuse some OPs from speech processing libraries. All these OPs are also included in `delta/layers/ops/py_x_ops.py` and can be used to either build the model graph or process data.

2.1.3 Numeric data

In addition to textual and acoustic features, the numeric feature (a.k.a., dense feature) is commonly used in machine learning. In fact, the numeric feature is a generic feature representation which can be directly fed into the model and most features are converted into numeric features for model training. In practise, a user may need to train a model using numeric features from different domains, e.g., click-through-rate, user profile, etc. We implement model training with numeric features in DELTA, which can be used in general machine learning tasks.

2.1.4 Multimodal data

We also want to highlight that DELTA supports multimodal training from different data sources. Multimodal training is very common in real applications. For example, in search engine, both textual features and user historic features (numeric features) are helpful for produce accurate search results. One may need to build a multimodal model to learn from textual features and numeric features simultaneously. Another example is that, in human speech emotion recognition, both acoustic features and textual features can be combined together to build a multimodal emotion recognizer (Xu et al., 2019).

To utilize multimodal data, there are many approaches to combine the features (Ngiam et al., 2011). In DELTA, we use direct concatenation to combine the features due to the simplicity. Our implementation includes multimodal training for textual+numeric features, acoustic+numeric features and textual+acoustic features. A user only need to define the training mode in the configuration file to start a multimodal training.

---

[3]https://www.tensorflow.org/api_docs/python/tf/data/Dataset
[4]https://github.com/fxsjy/jieba
[5]https://github.com/tensorflow/lingvo

2.2 Models

To build models in DELTA, we wrap the low-level neural network models as basic components, which can be directly used in high-level models. The basic components include multi-layer perceptrons (MLPs) (Rosenblatt, 1958; Rumelhart et al., 1988), convolutional neural networks (CNNs) (LeCun et al., 1990), long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997), attention mechanism (Bahdanau et al., 2015), transformers (Vaswani et al., 2017), etc. These components are widely used in NLP models and speech models for different tasks and they are implemented in `delta/layers` directory.

Note that, we have implemented the most popular high-level models in DELTA, meaning that a user can directly use the provided configuration files for model training. For most high-level models, our implementation refers to publications, and some modification may be applied. If you want to develop a new high-level model, you build a new model in `delta/models` with a python register.

2.2.1 NLP models

In DELTA, we organize the NLP models based on the input/output data format. For different NLP tasks, if the input and output data formats are the same, they usually can be addressed by similar high-level architecture. For example, in high-level, a sequence-to-sequence architecture can be used to address both machine translation and text summarization, with some modification for each task. We currently implement the following model architectures:

- *Sequence classification*
  This model deals with tasks like sentence classification and document classification. Here, the input is a sequence of text and the target is a label. For document classification, you can define a sentence separator to divide the documents into a sequence of sentences for hierarchical modeling. We provide the sequence classification models using CNN (Collobert et al., 2011; Kim, 2014), LSTM (Tai et al., 2015), hierarchical attention networks (Yang et al., 2016), transformer (Vaswani et al., 2017), etc.

- *Sequence labeling*
  For this model, both the input and the output are sequences and the model assigns a label to each step in the input sequence, so the input sequence and the output sequence should have the same length. This model is usually used in named entity recognition (NER), part-of-speech tagging, etc. We have provided the LSTM based sequence labeling (Chiu and Nichols, 2016) and an LSTM with a conditional random field (CRF) based method (Huang et al., 2015; Lample et al., 2016; Ma and Hovy, 2016).

- *Pairwise modeling*
  The input for this model is a pair of two documents. The model is trained to learn the relationship between them. In this model, each document is modeled separately to generate its document embedding, then the two embedding vectors are either concatenated together and fed into a classifier or compared to compute the cosine similarity. This type of models has been widely used on document similarity, query-based question-answering (QA), natural language inference (NLI), etc. The key for this approach is to supervisedly train the networks for document embedding (Bowman et al., 2015; Conneau et al., 2017; Huang et al., 2013).

- *Sequence-to-sequence (seq2seq) modeling*
  The seq2seq model takes input as a sequence of text and generate a sequence of text. But the lengths of two sequences are not necessarily the same. This structure usually has an encoder to learn the information from the input sequence and then uses a decoder to autoregressively generate the output. It has been widely used in many text generation tasks, for example, machine translation, text summarization, dialogue generation, etc. We implement the standard seq2seq models using LSTM with attention (Bahdanau et al., 2015) and transformers (Vaswani et al., 2017). Note that, the seq2seq structure is also used for speech recognition. In DELTA, this part is shared between NLP and ASR tasks.

- *Multitask modeling*
  Since the models in DELTA are modularized, it is easy to support multitask learning using the existing modules. As an example, in a dialogue system, an intent detection model and a slot filling model can be jointly trained using a neural network, which is a typical multitask learning task (Li et al., 2018; Liu and Lane, 2016). Based on the idea, we implement a

bidirectional LSTM with CRF for the joint modeling, where the intent detection is trained with the loss aggregated from the whole sequence and the slot filling is trained with the loss in each time step.

- *Pretraining integration*
  Recent NLP studies have been showing significant improvements on using pretrained language models on unlabeled data, e.g., ELMO (Peters et al., 2018), BERT (Devlin et al., 2019). We implement an interface to integrate a pretrained model into a DELTA model, where the pretrained model is used to dynamically generate embedding which is concatenated with the word embedding for the different task. To be specific, a user can pretrain an ELMO or BERT model first and then build a DELTA model with the prertained model. Both model will be combined into a TensorFlow graph for training and inference. The ELMO or BERT models trained from the official open-sourced libraries can be directly used in DELTA.

2.2.2 Speech models

For speech processing, the model usually couple with the feature extraction because different task may have different features extracted using signal processing. In order to better organize the model, we implement the speech models as a task-specific fashion.

- *Automatic speech recognition (ASR)*
  ASR is one of the most important task in speech processing, where the input is a raw speech waveform in time series and the output is the corresponding text. Recent trend in ASR is to build an end-to-end ASR system using seq2seq framework. We provide an attention based seq2seq ASR model (Bahdanau et al., 2016; Chan et al., 2016). We also implement another popular type of ASR model using connectionist temporal classification (CTC) (Graves et al., 2006).

- *Speaker verification*
  Speaker verification is the process of verifying whether an utterance belongs to a specific speaker, based on that speaker's known utterances (i.e., enrollment utterances). It has applications on voice match, identification check, etc. We provide an X-vector text-independent model (Snyder et al., 2018) and an end-to-end model (Wan et al., 2017).

- *Speech emotion recognition*
  From the machine learning perspective, speech emotion recognition is similar to speaker identification, because both are speech sentence classification. However, the features used in both tasks can be very different, therefore we provide both models separately. Recently several deep learning based approaches have been successfully used in speech emotion recognition (Han et al., 2014; Satt et al., 2017) and we implement some models the in DELTA.

2.2.3 Multimodal models

Multimodal modeling is paramount to practical applications because real data usually comes from different sources and available in multiple modalities. Since it is still a research topic to how to model multimodal data, we provide a straightforward approach to concatenate the multimodal data in the input feature stage for modeling (Ngiam et al., 2011). This is a framework to use multimodal data for general purpose. More sophisticated data fusion approaches are usually task dependent and we provide an example using aligned textual and acoustic features for emotion recognition (Xu et al., 2019).

- *Textual+acoustic*
  A common applications for multimodal learning with text and speech is speech emotion recognition, where the model takes input as speech signals and the corresponding transcripts and output the emotion states. In our implementation, we use two sequential models (e.g., CNNs or LSTMs) to learn the sequence embedding for speech and text separately, and then concatenates the learned embedding vectors for classification.

- *Textual+numeric*
  Multimodal learning with textual and numeric features was widely used in real application.

In DELTA, we implement the direct concatenation data fusion in data processing stage, therefore this type of multimodal training can be directly used for existing models in DELTA.

2.3 Training pipeline

In this subsection, we introduce the whole training pipeline in DELTA. With the existing models in DELTA, a user only need to prepare the dataset and provide a configuration file in the format of yaml. As shown in Fig. 2, the configuration contains the parameters used in model training. The task configuration indicates the model or task types including several NLP and speech tasks as well as the multimodal tasks. The model configuration defines the model parameters such as types of neural networks, number of hidden layers, number of neural units. The training configuration mainly specify the parameters for optimization engines and evaluation metrics. Once the configuration files are defined, you can simply start a training pipeline with one line of command.
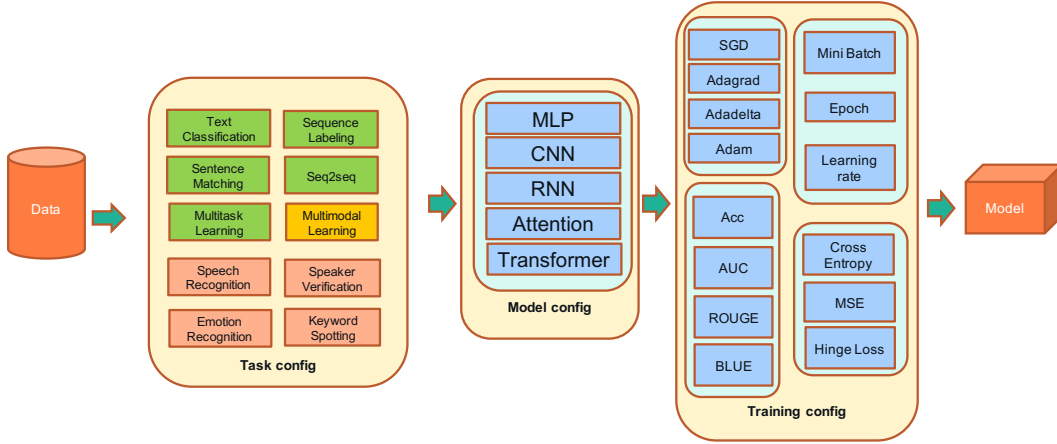

Figure 2: Configuration of model training in DELTA.

To deep dive into the training pipeline, we now discuss the detailed implementation methods in DETLA. From the aspect of code structure, a model training process uses a uniform training pipeline, which consists of three components: *task*, *model*, and *solver*.

2.3.1 Task

Task is an abstract class for data processing as mentioned in Section 2.1, including feature processing and data formatting. It applies online or offline data processing, and the processed data can be fed into model for training, evaluation, or inference. It includes several abstract methods for generating, batching, feature preprocessing and shuffling the data examples. After data processing, it wraps the data examples by `tf.data.dataset`, which supports several functions such as batching, padding, shuffling, multi-threading processing. This is similar to the design in Tensor2Tensor (Vaswani et al., 2018) and compatible with both TensorFlow 1.x and 2.x. The abstract methods in *Task* are brief described as follows:

- `generate_data` is a python generator to yield one example in every iteration.
- `feature_spec` is a specification of the *types* and *shapes* of the data example generated by `generate_data`.
- `preprocess_batch` is the main class for feature processing of batched examples. You can apply different feature processing in this class, such as *text tokenization*, *Chinese word segmentation*, *audio feature extraction*, *speech feature normalization*. Note that, in DELTA, each feature preprocessing operations have been implemented as a TensorFlow OP, therefore it is straightforward to use it as a model component and build an end-to-end model graph. To develop new feature processing OP, one can also program customized TensorFlow OPs in DELTA.
- `dataset` is the main class of data pipeline, and a wrapper for `tf.data.dataset`. All data processing of examples are organized in this class.

6

- `input_fn` generates a function without parameters, which is also recommended by Tensor-Flow Estimator[6].

For speech processing tasks, e.g., automatic speech recognition and speaker identification, the inputs are usually raw waveform files and you may need to apply extra offline feature processing, such as speech feature extraction and global feature normalization. These methods mainly follow the feature processing part in the speech recognition library Kaldi (Povey et al., 2011).

- `generate_feat` is responsible for generating speech feature from raw speech data, such as spectrogram feature or Mel-frequency cepstral coefficients (MFCCs).
- `generate_cmvn` compute global cepstral mean and variance normalization (CMVN) on generated speech feature.

2.3.2 Model building

For model building, to compatible with both TensorFlow 1.x and 2.x, we introduce two types of models in DELTA: *Keras model* and *raw model*. Keras (Chollet et al., 2015) is a popular high-level API for building deep learning models, which has been adopted into TensorFlow 2.x, where the high-level API is changed from `tf.layers` to `tf.keras.layers`. To build a new model in DELTA, we recommend using Keras API because it is user-friendly and easy to extend. To compatible with legacy system, DELTA also supports the raw model type as in TensorFlow 1.x, which can be deployed into production for TensorFlow 1.x system.

- *Keras model*
  Keras models supports object-oriented design, including variants, layers, models and checkpoint management. This is the default model type in DELTA and we use the subclass style paradigm of Keras to construct sublayers or submodels. In DETLA, a model is initialized in `__init__()` method and called in `call()` method.
- *Raw model*
  For the sake of compatibility, we also use raw models. The interface of raw models is similar to that of Keras models, so we use the unified paradigm for model building. We use `tf.variable_scope` to wrapper the scope of the model implementation.

2.3.3 Solver

A *Solver* is a module for composing the pipeline containing model construction, training and evaluation process. When building a model, you can choose a solver based on the model or task types. For example, you may want to choose `RawClassSolver` if the model you build is a subclass of `TextClassModel`. The most recommended solver base clase are `EstimatorSolver` and `RawSolver`. The main abstract methods of *Solver* are described as follows:

- `train()` builds a model for the training mode. This process include training data fetching, iteration on datasets, optimization, and parameters updating. The process is usually implemented with `tf.train.MonitoredTrainingSession`. This function saves the model to a TensorFlow checkpoint for every specified training steps.
- `eval()` builds the model for the evaluation mode, including evaluation data fetching with labels, inference, and measurement. The metrics are defined in `delta.utils.metrics`.
- `infer()` is similar to `eval()`, but it takes unlabeled data as inputs for inference. In addition, for some tasks, e.g., keyword spotting (Chen et al., 2014), we need to add post processing after the model inference, so `infer()` runs model inference followed by a `postproc_fn()` function.
- `train_and_eval()` is the combination of `train()` and `eval()`. It applies the evaluation for every specified training steps.
- `export_model()` transforms the model checkpoint saved during training process to a TensorFlow *SavedModel* format[7], which is supported by most deployment environments.

---

[6]https://www.tensorflow.org/guide/estimators
[7]https://www.tensorflow.org/guide/saved_model

3 Deployment

To our knowledge, most open-sourced deep learning platforms are designed for model training and do not focus on model serving. However, serving a model for industrial applications is not trivial. In order to facilitate productizing a trained model, we provide deployment features for model serving in DELTA.

The deployment part is referred as DELTA-NN and implementations are mainly organized under `deltnn` and `dpl` directories. Fig. 3 shows an overview of DELTA-NN. A trained model is optionally compressed or optimized in terms of computational efficiency. The deployment pipeline is able to convert the model into several formats as needed. As shown in the figure, TF Serving supports two interfaces, GRPC and RESTful API. They are widely used in online serving in production. For a bare metal environment, you can wrap DELTA-NN as a library with interfaces for model inference.
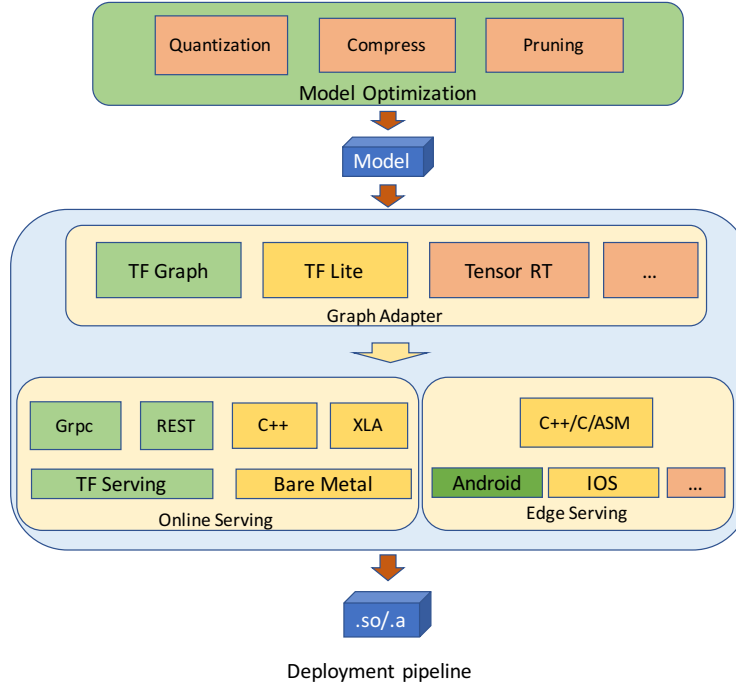

Figure 3: High-level overview of productization.

3.1 Model Optimization

Optimizing the computational efficiency is important to real applications. Deploying model into mobile or edge devices has restricted requirement on the models, such as limited memory and low power consumption. Some embedded devices do not have floating-point units and can only support integer accelerator. In DELTA-NN, we integrate a model distillation component which is used to learn a small model from an existing large model (Hinton et al., 2014). provide an interface for model optimization, such as model quantization, model pruning. We also

3.2 Model serving

For productization, DELTA-NN supports multi-graphs online and/or offline inference, with CPU, GPU and edge devices. There are several serving modes: TFModel, TFLite and TFServing. We can also combine these serving modes as a customized mode.

- *TFModel*
  This mode is provided by the standard TensorFlow library. TensorFlow provides several model formating, including CheckPoint, GraphDef, and SavedModel, etc. We recommend using SavedModel as the model format for standard model serving.

- *TFLite*
  TensorFlow Lite[8] is a lightweight solution for mobile and embedded devices for TensorFlow models. It enables low-latency inference of on-device machine learning models with a small binary size and fast performance supporting hardware acceleration. DELTA-NN supports using TensorFlow Lite as an engine for mobile and embedded devices.

- *TFServing*
  TensorFlow Serving[9] is a flexible, high-performance serving system for machine learning models, designed for production environments. It deals with the inference aspect of machine learning, taking models after training and managing their lifetimes, providing clients with versioned access via a high-performance, reference-counted look-up table. TensorFlow Serving provides out-of-the-box integration with TensorFlow models, and can be easily extended to serve other types of models. DELTA-NN implements an HTTP/HTTPS Client to request remote TensorFlow Serving model results. Combining with TFLite, you can easily deploy offline serving solutions.

3.3 Deployment pipeline

The deployment pipeline is organized as scripts and put into the folder `dpl`.

A user is responsible for providing TensorFlow *SavedModel* and a user-defined *model configuration*. Based on the configuration, DELTA-NN can automatically generate a deployment pipeline, including the following steps:

1. Model optimization
   This is a place-holder for model distillation, quantization, pruning, compression. You can choose to use a third-party library for model optimization.

2. Model graph transformation
   This step converts a *SavedModel* to a proper model format for serving on cloud, mobile and/or embedded devices, e.g., frozen graph, saved model, TFLite model, etc. Scripts are under `dpl/gadpter`.

3. Compiling library
   This step compiles the model inference engine library. It will compile models on different platforms and devices with the corresponding engine libraries, e.g., TensorFlow C++ library for CPU and GPU, TFLite library for Android and iOS, etc.

   After compiling the engine library, the step will compile DELTA-NN library. DELTA-NN abstracts low-level engine interface as a uniform interface, which supports multi-graphs inference, offline and online mixed inference, and multiple devices inference. These features are very useful in production.

   The engine library and DELTA-NN library are then combined into a composed binary for application.

4. Testing
   Testing is critical to an industrial level application. We release a Docker image[10] to perform unit test, integration test and system test. This process is automated through continuous integration (CI).

   A user can perform A/B test as part of CI. You can set up some metrics as trigger signals for continuous deployment when deploying a new model. We recommend using Docker to maintain the deployment process.

4   Benchmarks

In order to evaluate the performance of the released models in DELTA, we provide experimental results for each task on public datasets as benchmarks. Note that, the purpose for the experiments is to provide solid and reliable benchmarks, therefore most experiments use those commonly-used models

---

[8] https://www.tensorflow.org/lite
[9] https://github.com/tensorflow/serving
[10] https://www.docker.com/

rather than the state-of-the-art algorithms. For each task, we compare our model with a similar model chosen from a highly-cited publication.

In DELTA, we organize all experiment settings together with the data processing scripts in the directory `egs`, similar to the usage in Kaldi (Povey et al., 2011) or ESPnet (Watanabe et al., 2018). As a DELTA user, you can easily reproduce the experimental results by using the configure files under corresponding dataset directories. You can also extend the algorithms based on these models for better performance.

4.1 NLP benchmark

We present the comparison results of our implementation on NLP models. For each task, we choose a model from a popular reference as the baseline. We use DELTA to build the similar models as in the publications and use the same datasets and metrics for comparison. The baseline results are the number provided in the reference. Table 1 shows the comparison results on several different NLP tasks and our results are on a par with the baselines. For the multitask joint training task and the summarization task, our implementations use standard neural networks and do not adopt the structures proposed in the papers therefore small performance gaps are expected.

Table 1: Performances of NLP models

Task	Model	Dataset	Metric	DELTA	Baseline	Reference
Sequence classification						
Sentences	CNN	TREC [a]	Acc	92.2	91.2	Kim (2014)
Documents	HAN	Yahoo Answer [b]	Acc	75.1	75.8	Yang et al. (2016)
Sequence labeling						
NER	BLSTM-CRF	CoNLL2003 [c]	F1	84.6	84.7	Huang et al. (2015)
Multitask						
Intent	BLSTM-CRF	ATIS [d]	Acc	97.4	98.2	Liu and Lane (2016)
Slot filling			F1	95.2	95.9	
Pairwise modeling						
NLI	LSTM	SNLI [e]	Acc	80.7	80.6	Bowman et al. (2016)
Seq2seq						
Summarization	BLSTM	CNN/Daily Mail [f]	RougeL	27.3	28.1	See et al. (2017)
Pretrained model						
NER	ELMO	CoNLL2003	F1	92.2	92.2	Peters et al. (2018)
NER	BERT	CoNLL2003	F1	94.2	94.9	Devlin et al. (2019)

[a] https://trec.nist.gov/data.html
[b] https://webscope.sandbox.yahoo.com/#datasets
[c] https://www.clips.uantwerpen.be/conll2003/ner/
[d] https://catalog.ldc.upenn.edu/docs/LDC93S4B/corpus.html
[e] https://nlp.stanford.edu/projects/snli/
[f] https://github.com/deepmind/rc-data

4.2 Speech benchmark

We are working on benchmarks for speech tasks, and we will update as soon as available.

5 Conclusion

We present DELTA, a deep learning based NLP and speech processing platform. DELTA provides modularized components and simple pipelines for use and develop. It also contains many deployment supports for real production. We conducted experiments on public datasets to demonstrate the reliable and solid implementation.

For future work, we will implement more advanced models and present the state-of-the-art results for NLP and speech tasks using DELTA. We also interested in the implementation of parallel model training, automatic parameter tuning, production integration, etc. More importantly, DELTA is an

open-sourced library and the contribution from the researchers and engineers in the community is highly appreciated.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations (ICLR 2015)*, 2015.

Dzmitry Bahdanau, Jan Chorowski, Dmitriy Serdyuk, Philemon Brakel, and Yoshua Bengio. End-to-end attention-based large vocabulary speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4945–4949. IEEE, 2016.

Samuel Bowman, Raghav Gupta, Jon Gauthier, Christopher D Manning, Abhinav Rastogi, and Christopher Potts. A fast unified model for parsing and sentence understanding. In *Proceedings of Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1466–1477, 2016.

Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2015)*, pages 632–642, 2015.

William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *Proceedings of the Conference on IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2016)*, pages 4960–4964. IEEE, 2016.

Guoguo Chen, Carolina Parada, and Georg Heigold. Small-footprint keyword spotting using deep neural networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2014)*, pages 4087–4091, 05 2014.

Jason PC Chiu and Eric Nichols. Named entity recognition with bidirectional lstm-cnns. *Transactions of the Association for Computational Linguistics*, 4:357–370, 2016.

François Chollet et al. Keras. https://keras.io, 2015.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12 (Aug):2493–2537, 2011.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP 2017)*, pages 670–680, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186, 2019.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the International Conference on Machine Learning (ICML 2006)*, pages 369–376. ACM, 2006.

Kun Han, Dong Yu, and Ivan Tashev. Speech emotion recognition using deep neural network and extreme learning machine. In *Proceedings of the international speech communication association (INTERSPEECH 2014)*, 2014.

Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29, 2012.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *Deep Learning Workshop of the Conference on Neural Information Processing Systems*, 2014.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997.

Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the ACM International Conference on Information & Knowledge Management (CIKM 2013)*, pages 2333–2338. ACM, 2013.

Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional LSTM-CRF models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, 2014.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS 2012)*, pages 1097–1105, 2012.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT 2016)*, pages 260–270, 2016.

Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS 1990)*, pages 396–404, 1990.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

Changliang Li, Liang Li, and Ji Qi. A self-attentive model with gate mechanism for spoken language understanding. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP 2018)*, pages 3824–3833, 2018.

Bing Liu and Ian Lane. Attention-based recurrent neural network models for joint intent detection and slot filling. In *Proceedings of the Conference on International Speech Communication Association (INTERSPEECH 2016)*, pages 685–689, 2016.

Xuezhe Ma and Eduard Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *Proceedings of the Conference on Annual Meeting of the Association for Computational Linguistics (ACL 2016)*, 2016.

Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. Multimodal deep learning. In *Proceedings of the International Conference on Machine Learning (ICML 2011)*, pages 689–696, 2011.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2018.

Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. The kaldi speech recognition toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, December 2011.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

Aharon Satt, Shai Rozenberg, and Ron Hoory. Efficient emotion recognition from speech using deep learning on spectrograms. In *Proceedings of the Conference on International Speech Communication Association (INTERSPEECH 2017)*, pages 1089–1093, 2017.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

Abigail See, Peter J Liu, and Christopher D Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1073–1083, 2017.

Jonathan Shen, Patrick Nguyen, Yonghui Wu, Zhifeng Chen, et al. Lingvo: a modular and scalable framework for sequence-to-sequence modeling, 2019.

D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, and S. Khudanpur. X-vectors: Robust dnn embeddings for speaker recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2018)*. IEEE, 2018.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS 2014)*, pages 3104–3112, 2014.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (ACL-IJCNLP 2015)*, volume 1, pages 1556–1566, 2015.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS 2017)*, pages 5998–6008, 2017.

Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan N. Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, Ryan Sepassi, Noam Shazeer, and Jakob Uszkoreit. Tensor2tensor for neural machine translation. *CoRR*, abs/1803.07416, 2018. URL http://arxiv.org/abs/1803.07416.

Li Wan, Quan Wang, Alan Papir, and Ignacio Lopez Moreno. Generalized end-to-end loss for speaker verification, 2017. URL https://arxiv.org/abs/1710.10467.

Shinji Watanabe, Takaaki Hori, Shigeki Karita, Tomoki Hayashi, Jiro Nishitoba, Yuya Unno, Nelson Enrique Yalta Soplin, Jahn Heymann, Matthew Wiesner, Nanxin Chen, Adithya Renduchintala, and Tsubasa Ochiai. Espnet: End-to-end speech processing toolkit. In *Proceedings of the Conference on International Speech Communication Association (INTERSPEECH 2018)*, pages 2207–2211, 2018.

Haiyang Xu, Hui Zhang, Kun Han, Yun Wang, Yiping Peng, and Xiangang Li. Learning alignment for multimodal emotion recognition from speech. In *Proceedings of the Conference on International Speech Communication Association (INTERSPEECH 2019)*, 2019.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2016*, pages 1480–1489, 2016.