DELTA

Release 1.0

Jul 16, 2020

Introduction

CHAPTER 1

DELTA - A DEep learning Language Technology plAtform

1.1 What is DELTA?

DELTA is a deep learning based end-to-end **natural language and speech processing** platform. 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. DELTA is mainly implemented using TensorFlow and Python 3.

For details of DELTA, please refer to this paper.

1.2 What can DELTA do?

DELTA has been used for developing several state-of-the-art algorithms for publications and delivering real production to serve millions of users. It helps you to train, develop, and deploy NLP and/or speech models, featuring:

- Easy-to-use
  - One command to train NLP and speech models, including:
    - NLP: text classification, named entity recognition, question and answering, text summarization, etc
    - Speech: speech recognition, speaker verification, emotion recognition, etc
  - Use configuration files to easily tune parameters and network structures
- Easy-to-deploy
  - What you see in training is what you get in serving: all data processing and features extraction are integrated into a model graph
  - Uniform I/O interfaces and no changes for new models
- Easy-to-develop
  - Easily build state-of-the-art models using modularized components
  - All modules are reliable and fully-tested

1.3 References

Please cite this paper when referencing DELTA.

```
@ARTICLE{delta,
       author = {{Han}, Kun and {Chen}, Junwen and {Zhang}, Hui and {Xu}, Haiyang and
         {Peng}, Yiping and {Wang}, Yun and {Ding}, Ning and {Deng}, Hui and
         {Gao}, Yonghu and {Guo}, Tingwei and {Zhang}, Yi and {He}, Yahao and
         {Ma}, Baochang and {Zhou}, Yulong and {Zhang}, Kangli and {Liu}, Chao and
         {Lyu}, Ying and {Wang}, Chenxi and {Gong}, Cheng and {Wang}, Yunbo and
         {Zou}, Wei and {Song}, Hui and {Li}, Xiangang},
       title = "{DELTA: A DEep learning based Language Technology plAtform}",
       journal = {arXiv e-prints},
       year = "2019",
       url = {https://arxiv.org/abs/1908.01853},
}
```

CHAPTER 2

Pick a installation way for yourself

2.1 Multiple installation ways

Currently we support multiple ways to install `DELTA`. Please choose one installation for yourself according to your usage and needs.

2.2 Install by pip

For the **quick demo of the features** and **pure NLP users**, you can install the `nlp` version of `DELTA` by pip with a simple command:

```
pip install delta-nlp
```

Check here for the tutorial for usage of delta-nlp.

Requirements: You need `tensorflow==2.0.0` and `python==3.6` in MacOS or Linux.

2.3 Install from the source code

For users who need **whole function of delta** (including speech and nlp), you can clone our repository and install from the source code.

Please follow the steps here: Install from the source code

2.4 Use docker

For users who are **capable of use docker**, you can pull our images directly. This maybe the best choice for docker users.

Please follow the steps here: Installation using Docker

CHAPTER 3

Install from the source code

To install from the source code, We use conda to install required packages. Please install conda if you do not have it in your system.

Also, we provide two options to install DELTA, `nlp` version or `full` version. `nlp` version needs minimal requirements and only installs NLP related packages:

```
# Run the installation script for NLP version, with CPU or GPU.
cd tools
./install/install-delta.sh nlp [cpu|gpu]
```

Note: Users from mainland China may need to set up conda mirror sources, see ./tools/install/install-delta.sh for details.

If you want to use both NLP and speech packages, you can install the `full` version. The full version needs Kaldi library, which can be pre-installed or installed using our installation script.

```
cd tools
# If you have installed Kaldi
KALDI=/your/path/to/Kaldi ./install/install-delta.sh full [cpu|gpu]
# If you have not installed Kaldi, use the following command
# ./install/install-delta.sh full [cpu|gpu]
```

To verify the installation, run:

```
# Activate conda environment
conda activate delta-py3.6-tf2.0.0
# Or use the following command if your conda version is < 4.6
# source activate delta-py3.6-tf2.0.0

# Add DELTA enviornment
source env.sh

# Generate mock data for text classification.
pushd egs/mock_text_cls_data/text_cls/v1
./run.sh
```

(continues on next page)

(continued from previous page)

```
popd

# Train the model
python3 delta/main.py --cmd train_and_eval --config egs/mock_text_cls_data/text_cls/
↪v1/config/han-cls.yml
```

CHAPTER 4

Installation using Docker

You can directly pull the pre-build docker images for DELTA and DELTANN. We have created the following docker images:

- delta-gpu-py3
- delta-cpu-py3
- deltann-gpu-py3
- deltann-cpu-py3

4.1 Install Docker

Make sure `docker` has been installed. You can refer to the official tutorial.

4.2 Pull Docker Image

You can build DETLA or DETLANN locally as *Build Images*, or using pre-build images as belows:

All avaible image tags list in here, please choose one as needed.

If you choose `delta-cpu-py3`, then download the image as below:

```
docker pull zh794390558/delta:delta-cpu-py3
```

4.3 Create Container

After the image downloaded, create a container.

For **delta** usage (model development):

```
cd /path/to/detla && docker run -v `pwd`:/delta -it zh794390558/delta:delta-cpu-py3 /
↪bin/bash
```

The basic version of **delta** (except Kaldi) was already installed in this container. You can develop in this container like:

```
# Add DELTA enviornment
source env.sh

# Generate mock data for text classification.
pushd egs/mock_text_cls_data/text_cls/v1
./run.sh
popd

# Train the model
python3 delta/main.py --cmd train_and_eval --config egs/mock_text_cls_data/text_cls/
↪v1/config/han-cls.yml
```

For **deltann** usage (model deployment):

```
cd /path/to/detla
WORKSPACE=$PWD
docker run -it -v $WORKSPACE:$WORKSPACE zh794390558/delta:deltann-cpu-py3 /bin/bash
```

We recommend using a high-end machine to develop DELTANN, since it needs to compile `Tensorflow` which is time-consuming.

CHAPTER 5

Manual Setup

This project has been fully tested on `Python 3.6.8` and `TensorFlow 2.0.0` under `Ubuntu 18.04.2 LTS`. We recommend that users use `Docker` or a virtual environment such as `conda` to install the python requirements.

5.1 Conda Package Install

5.1.1 Build conda envs

```
conda create -p <path>/<env_name> python=3.6
source activate <path>/<env_name>
```

5.1.2 Install Tensorflow

```
conda install tensorflow-gpu=2.0.0
```

5.1.3 Install dependences

Delta dependient on third party tools, so when run the program, need blow to install tools:

activate the environment and use below

```
cd tools && make
```

5.2 Pip Install

For case you want install `Tensorflow Gpu 2.0.0`, under machine which has `Gpu Driver 410.48`. It has problem of runtime not compariable with driver version, when isntall using *conda*. Then we can install tensorflow

from `Pip` as below:

5.2.1 Build conda envs

Same to *conda install*.

5.2.2 Install CUDA toolkit and CUDANN

See CUDA Compatibility for `CUDA Toolkit and Compatible Driver Version`. See cuDNN Support Matrix for `cuDNN For CUDA and NVIDIA Hardware`.

For Nvidia Driver Version: 418.67, CUDA Version: 10.1:

```
conda install cudatoolkit==10.1.168-0
conda install cupti=10.1.168-0
conda install cudnn==7.6.0
```

or

```
conda install cudatoolkit==10.1
conda install cupti==10.1
conda install cudnn==7.6.0
```

For user in China, we can set conda mirror as below:

```
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free/
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main/
conda config --set show_channel_urls yes
```

Other references: conda-forge tuna

5.2.3 Install Tensorflow

```
pip install  -i https://pypi.tuna.tsinghua.edu.cn/simple tensorflow-gpu==2.0.0
```

For tensorflow 2.0.0, make sure numpy version is 1.16.4.

5.2.4 Install dependences

Same to *conda install*.

5.3 DELTA install

5.3.1 NLP User

Install DELTA without `speech` dependences:

```
cd tools && make basic check_install
```

5.3.2 Speech User

By default we will install DELTA with `Kaldi` toolkit:

```
cd tools && make delta
```

If user has installed `Kaldi`, please DELTA as below:

```
cd tools && make delta KALDI=<kaldi-path>
```

it is simply link the `<kaldi-path>` to `tools/kaldi`.

5.3.3 Advanced User

Please see `delta` target of `tools/Makefile`.

5.4 DELTANN install

Install DELTANN as below:

```
cd tools && make deltann
```

For more details, please see `deltann` target of `tools/Makefile`

CHAPTER 6

Install on macOS

Running DELTA training on a macOS is mostly the same as running on Linux, except some minor differences.

6.1 Python environment

You need to set up a working Python 3.6.x environment, either by using conda or manually build from source. You can follow the instructions in `manual_setup.md` to set up python and the required packages, e.g. Tensorflow. Note: `tensorflow-gpu` requires nvidia GPU, which might not be supported the latest macOS versions. You may want to use the `tensorflow` package (no -gpu postfix) instead. Some models that uses cuDNN implementations will not work without a CUDA GPU however.

6.2 Other requirements

6.2.1 Notes for Kaldi

Building and running Kaldi on a macOS requires `wget`, `gawk` and other utilities which need to be installed via `Homebrew`. See `https://brew.sh` for details.

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
↪master/install)"
brew install wget gawk grep
```

Also the `mmseg` package for Python2 is needed:

```
pip2 install mmseg
```

Then follow `manual_setup.md` / `DELTA install` section to install 3rd-party dependencies.

CHAPTER 7

Reproduce experiments - egs

The `egs` director is data-oriented for `data prepration` and model `training`, `evaluation` and `infering`.

Sppech and NLP task are orgnized by `egs`, e.g. ASR, speaker verfication, NLP.

7.1 An Egs Example

In this tutorial, we demonstrate an emotion recognition task with an open source dataset: `IEMOCAP`. All other task is same to this.

A complete process contains following steps:

- Download the `IEMOCAP` corpus.
- Run egs/iemocap/emo/v1/run.sh script

Before doing any these steps, please make sure that `delta` has been successfully installed.

Every time you re-open a terminal, don't forget:

```
source env.sh
```

7.1.1 Prepare the Data Set

Download `IEMOCAP` from https://sail.usc.edu/iemocap/index.html

7.1.2 Run

First:

```
pushd egs/iemocap/emo/v1
```

Then run `run.sh` script

```
./run.sh --iemocap_root=</path/to/iemocap>
```

For other task, e.g. `ASR`, `Speaker`, the main script is `run_delta.sh`, but default main root is `run.sh`.

CHAPTER 8

Speech Features

8.1 Goal

Add custom speech feature extraction ops, and compare the extracted features with kaldi's.

8.2 Procedure

1. Create custom C++ op, **'xxx.h'** and **'xxx.cc'**

   Files should be stored in **delta/layers/ops/kernels/**, details can refer to existing files, e.g., pitch.cc / pitch.h

2. Implement the kernel for the op, **'xxx_op.cc'**

   Files should be stored in **delta/layers/ops/kernels/**, details can be found in Tensorflow Guild: Adding a New Op

3. Define the op's interface, **'x_ops.cc'**

   Files should be stored in **delta/layers/ops/kernels/**, details in above link

4. Compile by using **'delta/layers/ops/Makefile'**
5. Register op in **'delta/layers/ops/py_x_ops.py'**
6. Unit-test **'xxx_op_test.py'**

8.3 Code Style

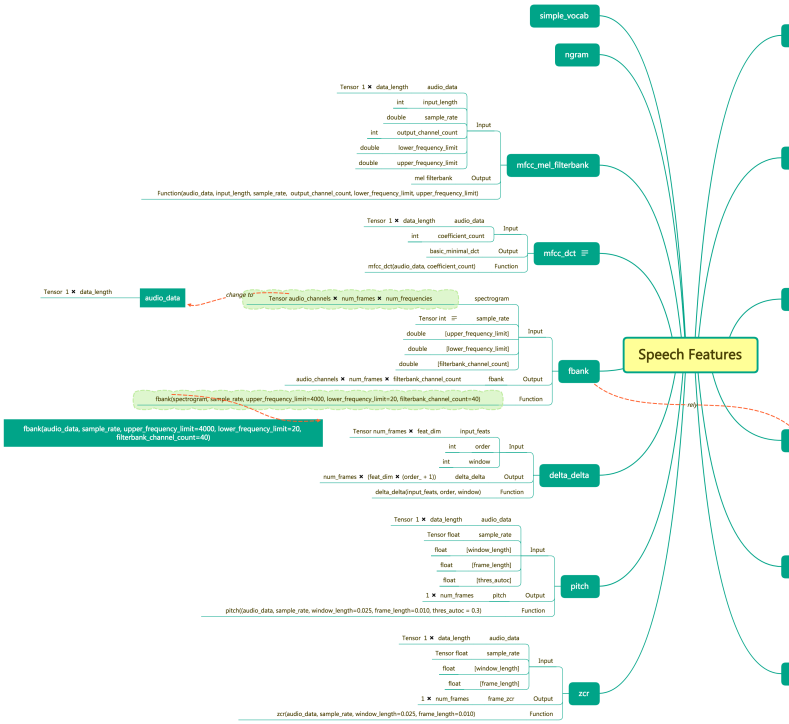
C++ code: using **clang-format** and **cpplint** for formatting and checking

Python code: using **yapf** and **pylint** for formatting and checking

Please follow Contributing Guide

8.4 Existing Ops

- Pitch
- Frame power
- Zero-cross rate
- Power spectrum (PS) / log PS
- Cepstrum / MFCC
- Perceptual Linear Prediction (PLP)
- Analysis filter bank (AFB)*Currently support window_length = 30ms and frame_length = 10ms for perfect reconstruction.*
- Synthesis filter bank (SFB)

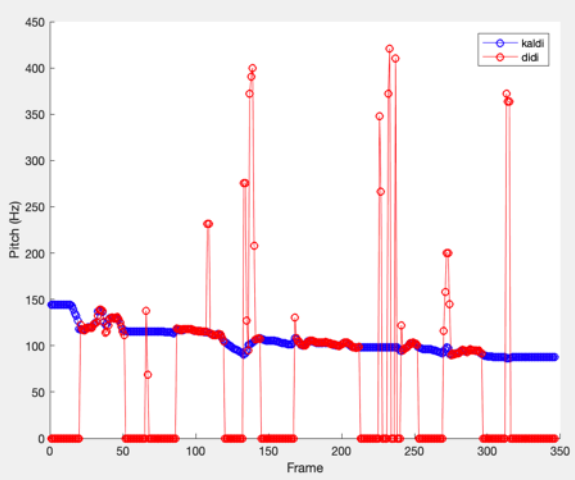


The specific interfaces of feature functions are shown below
Features

8.5 Comparsion with KALDI

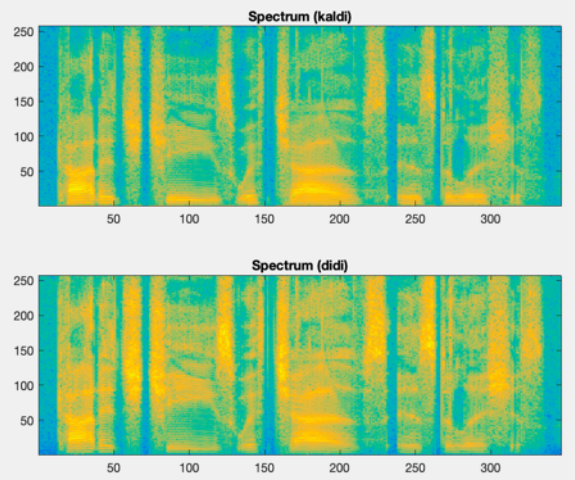
Extracted features are compared to existing KALDI features.

1. Pitch



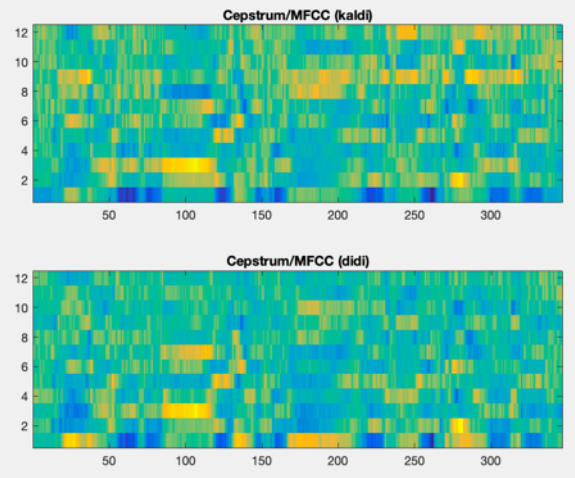
Pitch

2. Log power spectrum



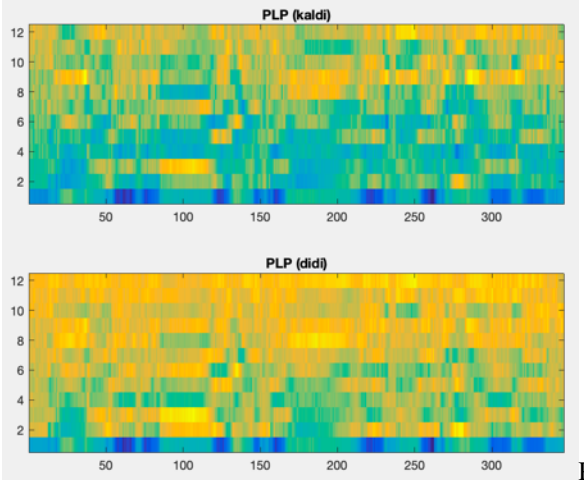
Log power spectrum

3. Cepstrum / MFCC



Cepstrum / MFCC

4. PLP



PLP

8.6 Reference

- An Intuitive Discrete Fourier Transform Tutorial
- Mel Frequency Cepstral Coefficient (MFCC) tutorial
- A Tutorial on Cepstrum and LPCCs
- Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between

CHAPTER 9

A Text Classification Usage Example for pip users

9.1 Intro

In this tutorial, we demonstrate a text classification task with a demo mock dataset **for users install by pip**.

A complete process contains following steps:

- Prepare the data set.
- Develop custom modules (optional).
- Set the config file.
- Train a model.
- Export a model

Please clone our demo repository:

```
git clone --depth 1 https://github.com/applenob/delta_demo.git
cd ./delta_demo
```

9.2 A quick review for installation

If you haven't install `delta-nlp`, please:

```
pip install delta-nlp
```

Requirements: You need `tensorflow==2.0.0` and `python==3.6` in MacOS or Linux.

9.3 Prepare the Data Set

run the script:

```
./gen_data.sh
```

The generated data are in directory: `data`.

The generated data for text classification should be in the standard format for text classification, which is "label\tdocument".

9.4 Develop custom modules (optional)

Please make sure we don't have modules you need before you decide to develop your own modules.

```
@registers.model.register
class TestHierarchicalAttentionModel(HierarchicalModel):
  """Hierarchical text classification model with attention."""

  def __init__(self, config, **kwargs):
    super().__init__(config, **kwargs)

    logging.info("Initialize HierarchicalAttentionModel...")

    self.vocab_size = config['data']['vocab_size']
    self.num_classes = config['data']['task']['classes']['num_classes']
    self.use_true_length = config['model'].get('use_true_length', False)
    if self.use_true_length:
      self.split_token = config['data']['split_token']
    self.padding_token = utils.PAD_IDX
```

You need to register this module file path in the config file `config/han-cls.yml` (relative to the current work directory).

```
custom_modules:
  - "test_model.py"
```

9.5 Set the Config File

The config file of this example is `config/han-cls.yml`

In the config file, we set the task to be `TextClsTask` and the model to be `TestHierarchicalAttentionModel`.

9.5.1 Config Details

The config is composed by 3 parts: `data`, `model`, `solver`.

Data related configs are under `data`. You can set the data path (including training set, dev set and test set). The data process configs can also be found here (mainly under `task`). For example, we set `use_dense: false` since no dense input was used here. We set `language: chinese` since it's a Chinese text.

Model parameters are under `model`. The most important config here is `name: TestHierarchicalAttentionModel`, which specifies the model to use. Detail structure configs are under `net->structure`. Here, the `max_sen_len` is 32 and `max_doc_len` is 32.

The configs under `solver` are used by solver class, including training optimizer, evaluation metrics and checkpoint saver. Here the class is `RawSolver`.

9.6 Train a Model

After setting the config file, you are ready to train a model.

```
delta --cmd train_and_eval --config config/han-cls.yml
```

The argument `cmd` tells the platform to train a model and also evaluate the dev set during the training process.

After enough steps of training, you would find the model checkpoints have been saved to the directory set by `saver->model_path`, which is `exp/han-cls/ckpt` in this case.

9.7 Export a Model

If you would like to export a specific checkpoint to be exported, please set `infer_model_path` in config file. Otherwise, platform will simply find the newest checkpoint under the directory set by `saver->model_path`.

```
delta --cmd export_model --config/han-cls.yml
```

The exported models are in the directory set by config `service->model_path`, which is `exp/han-cls/service` here.

CHAPTER 10

A Text Classification Usage Example

10.1 Intro

In this tutorial, we demonstrate a text classification task with an open source dataset: `yahoo answer` for users with installation from source code..

A complete process contains following steps:

- Prepare the data set.
- Set the config file.
- Train a model.
- Export a model
- Deploy the model.

Before doing any these steps, please make sure that `delta` has been successfully installed.

Every time you re-open a terminal, don't forget:

```
source env.sh
```

10.2 Prepare the Data Set

You can refer to directory: `egs` for data preparing. In our example, `egs/yahoo_answer` contains data preparing including downloading and reformat.

First:

```
cd egs/yahoo_answer/text_cls/v1
```

Then run the script:

```
./run.sh
```

The generated data are in directory: `data/yahoo_answer`.

The generated data for text classification should be in the standard format for text classification, which is "label\tdocument".

10.3 Set the Config File

The config file of this example is `egs/yahoo_answer/text_cls/v1/config/cnn-cls.yml`

In the config file, we set the task to be `TextClsTask` and the model to be `HierarchicalAttentionModel`.

10.3.1 Config Details

The config is composed by 3 parts: `data`, `model`, `solver`.

Data related configs are under `data`. You can set the data path (including training set, dev set and test set). The data process configs can also be found here (mainly under `task`). For example, we set `use_dense: false` since no dense input was used here. We set `language: chinese` since it's a Chinese text.

Model parameters are under `model`. The most important config here is `name: SeqclassCNNModel`, which specifies the model to use. Detail structure configs are under `net->structure`. Here, the `filter_sizes` are 3, 4, 5 and `num_filters` is 128.

The configs under `solver` are used by solver class, including training optimizer, evaluation metrics and checkpoint saver. Here the class is `RawSolver`.

10.4 Train a Model

After setting the config file, you are ready to train a model.

```
python delta/main.py --cmd train_and_eval --config egs/yahoo_answer/text_cls/v1/
↪config/cnn-cls.yml
```

The argument `cmd` tells the platform to train a model and also evaluate the dev set during the training process.

After enough steps of training, you would find the model checkpoints have been saved to the directory set by `saver->model_path`, which is `exp/yahoo_answer/ckpt/cnn-cls` in this case.

10.5 Export a Model

If you would like to export a specific checkpoint to be exported, please set `infer_model_path` in config file. Otherwise, platform will simply find the newest checkpoint under the directory set by `saver->model_path`.

```
python delta/main.py --cmd export_model --config egs/yahoo_answer/text_cls/v1/config/
↪cnn-cls.yml
```

The exported models are in the directory set by config `service->model_path`, which is `exp/yahoo_answer/cnn-cls/service` here.

10.6 Deploy the Model

Before model deploying, please make sure that `deltann` has been successfully installed.

CHAPTER 11

ASR Data

This tutorials discusses how to deal with automatic speech recognition(ASR) tasks on the basis of DELTA.

11.1 Data description

For data preparing, you can refer to directory:'egs/hkust/asr/v1'. By simply using `./run.sh`, an open source dataset, HKUST, can be quickly downloaded and reformated like below:

```
uttID: {
   "input": [
      {
        "feat": the file and the position while the feats of current utterance is␣
↪sorted
        "name": "input1"
        "shape" : [
            number_frames
            dimension_feats
        ]
      }
   ],
   "output": [
       {
           "name": "target",
           "shape": [
               number_words
               number_classes
           ],
           "text":
           "token":
           "tokenid":
       }
   ],
```

(continues on next page)

(continued from previous page)

```
   "utt2spk": speaker index
}
```

It should be noted that num_classes = size_vocabulary + 2, where size_vocabulary is the size of the vocabulary. The zero value and the largest value (num_classes - 1) is reserved for the blank and sos/eos label respectively. For Example, the vocabulary is consist of 3 different labels [a, b, c]. Then, num_classes = 5 and the labels indexing is {blank:0, a:1, b:2, c:3, sos/eos:4}

11.2 Model training

1. For ASR tasks, a default config file is written in `conf/asr-ctc.yml`. Two different CTC-based model, `CTCAsrModel` and `CTC5BlstmAsrModel`, are supported in DELTA. The details of them can be seen in `delta/models/asr_model.py`.
2. After setting the config file, the following script can be executed to train a ASR model:

```
python3 delta/main.py --config egs/hkust/asr/v1/conf/asr-ctc.yml --cmd train_and_eval
```

1. Same as the Espnet, the class index of blank label is set to be 0 in AsrSeqTask. However, the default blank label used in Tensorflow.nn.ctc_loss is num_classes - 1. To solve this problem, the `ctc_data_transform` interface is supported in `delta/utils/loss/loss_utils.py`. For logits generated by the ASR model, this interfance moves the blank_label cloumn to the end of it. For input labels, this interface changes the value of blank_label elements to num_classes - 1, and the value of other labels whose class index is greater than blank_label is reduced by 1.
2. In `delta/utils/decode/tf_ctc.py`, two different methods, `ctc_greedy_decode` and `ctc_beam_search_decode`, are supported to perform greedy and beam search decoding on the logits respectively. In this stage, the mismatch between the blank label index in input logits and num_classes - 1 could also occur. Thus, we provide the `ctc_decode_blankid_to_last` method to address this issue. Specially, in order to eliminate the effect of the change of blank label index, the `ctc_decode_last_to_blankid` should be applied on the decode result which removing repeated labels and blank symbols to adjust the index of blank label back.

CHAPTER 12

Deployment scripts - dpl

The `dpl` directory is for orginating model `config`, `convert`, `testing`, `benchmarking` and `serving`.

12.1 Inputs & Outputs

After model is exported as `SavedModel`, we recommend using Netron to view the neural network model, then getting the `inputs` and `outputs` names.

Other tools to determine the inputs/outputs for GraphsDef protocol buffer:

- summarize_graph
- TensorBoard To visualize a .pb file, use the import_pb_to_tensorboard.py script like below:

```
python import_pb_to_tensorboard.py --model_dir <model path> --log_dir <log dir path>
```

- TFLite Run the visualize.py script with bazel:

```
bazel run //tensorflow/lite/tools:visualize model.tflite visualized_model.html
```

12.2 Model directory

Putting the `SavedModel` under `dpl/model` directory, config the `dpl/model/model.yaml` as it is.

12.3 Graph Convert

Running `dpl/gadpter/run.sh` to convert model to other model format, e.g. `tflite`, `tftrt`, `ngraph`, `onnix`, `coreml` and so on.

12.4 Build Packages

All packges build under `docker` env, see `docker/dpl`.

- build tensorflow cpu
- build tensorflow gpu
- build tensorflow with TensorRT
- build tensorflow lite cpu
- build tensorflow lite Android
- build tensorflow lite IOS
- build DELTA-NN with dependent packages
- build unit-test
- build examples under DELTA-NN

12.5 Testing

Do belows testing under `docker` env, if all passed, then deployment the model:

- unit testing
- integration testing
- smoke testing
- stress testing

12.6 AB Testing

If model is better than old model by `metrics` and `RTF`, then we push it to Could or Edge.

12.7 Deployment

12.7.1 Deploy Mode

For Could, deployment as belows mode:

1. DELTA-NN Serving
   - DELTA-NN TF CPU
   - DELTA-NN TFTRT GPU
   - DELTA-NN Client
1. DELTA-NN TF-Serving

For Edge, as:

- DELTA-NN TFLite
- DELTA-NN Client

12.7.2 Deploy Env

For Could, pack library, bin and model into `docker`, then using `K8s+docker` to depoyment. For Edge, pack library, bin and model as `tarball`.

CHAPTER 13

DELTA-NN architecture

1. features
2. compile
3. package

- TF-Serving
- Serving
- Embedding
- Client

13.1 Features

- tiny size
- pack all by docker
- supporting custom-op
- exporting only C api
- compatibility TF-Serving and Serving RESTful API
- compatibility with Could and Edge usage
- supporting multi graphs inference
- supporting multi modal application, e.g KWS(Edge) + ASR(Could)

13.2 Compile

How to compile `delta-nn`.

13.3 TF-Serving

13.3.1 Suport Custom-Op

Compile TF-Serving with custom Ops.

13.3.2 Serving with Docker

- Tensorflow Serving with Docker
- Serving with Docker using your GPU
- Tensorflow Serving with Kubernetes

13.4 Serving

Using `go` to wraper DELTA-NN supporting HTTP/HTTPS protocal.

13.4.1 Support Engines

- TF
- TFTRT

13.4.2 REST API

Compatibility with TF-Serving RESTful API.

- input tensors in row format
- input tensors in column format

13.4.3 Data Exchange

Using json to exchange data with Client and Serving, since it support CBOR and BSON specification.

13.5 Embedding

13.5.1 Support Engines

- TFLite

13.5.2 Core

DELTA-NN using TFLite as backend engine.

13.6 Client

13.6.1 Support Engines

- TF
- TFTRT
- TFLite

13.6.2 HTTP/HTTPS Client

DELTA-NN using mbedtls as SSL/TLS library.

It's an OpenSSL alternative libraray, which has many features:

- Fully featured SSL/TLS and cryptography library
- Easy integration with a small memory footprint
- Easy to understand and use with a clean API
- Easy to reduce and expand the code
- Easy to build with no external dependencies
- Extremely portable

CHAPTER 14

Develop with Docker

14.1 Install Docker

Make sure `docker` has been installed. You can refer to the official tutorial.

14.2 Development with Docker

You can build DETLA or DETLANN locally as *Build Images*, or using pre-build images as belows:

All avaible image tags list in here, please choose one as needed.

If we choose `delta-cpu-py3`, then download the image as below:

```
docker pull zh794390558/delta:delta-cpu-py3
```

After the image downloaded, create a container:

```
cd /path/to/detla && docker run -it -v $PWD:/delta zh794390558/delta:delta-cpu-py3 /
↪bin/bash
```

then develop as usual.

We recommend using a power machine to develop DELTANN, since it needs to compile `Tensorflow` which is time-consuming.

14.2.1 Tags

Please go to this to see the valid docker images tags.

14.3 Build Images

14.3.1 Build CI Image

```
pushd docker && bash build.sh ci cpu build && popd
```

14.3.2 Build DELTA Image

For building cpu image:

```
pushd docker && bash build.sh delta cpu build && popd
```

for building gpu image

```
pushd docker && bash build.sh delta gpu build && popd
```

14.3.3 Build DELTANN Image

For building cpu image:

```
pushd docker && bash build.sh deltann cpu build && popd
```

for building gpu image

```
pushd docker && bash build.sh deltann gpu build && popd
```

CHAPTER 15

DELTA-NN compile

Deltann support tensorflow tensorflow liteand tensorflow serving.

15.1 Tensorflow C++

Build tensorflow for Linux :

1. Install under deltann docker.

```
cd tools/ && ./install/install-deltann.sh
```

1. Config tensorflow build

```
cd tools/tensorflow
```

Configure your system build by running the ./configure,

1. Build tensoflow library

15.1.1 CPU-only

```
bazel build -c opt --verbose_failures //tensorflow:libtensorflow_cc.so
```

mkl support

```
bazel build -c opt --config=mkl --verbose_failures //tensorflow:libtensorflow_cc.so
```

15.1.2 GPU support

Configure your system build by running the ./configure.

For GPU support, set cuda=Y during configuration and specify the versions of CUDA and cuDNN.

```
Do you wish to build TensorFlow with CUDA support? [y/N]: y
CUDA support will be enabled for TensorFlow.

Please specify the CUDA SDK version you want to use. [Leave empty to default to CUDA␣
↪9.0]: 10


Please specify the location where CUDA 10.0 toolkit is installed. Refer to README.md␣
↪for more details. [Default is /usr/local/cuda]:


Please specify the cuDNN version you want to use. [Leave empty to default to cuDNN 7]:


Please specify the location where cuDNN 7 library is installed. Refer to README.md␣
↪for more details. [Default is /usr/local/cuda]:
```

Build

```
bazel build -c opt --config=cuda --verbose_failures //tensorflow:libtensorflow_cc.so
```

15.1.3 Tensoflow TensorRT support

Configure your system build by running the ./configure. For TensorRT support, set Y during configuration and specify the versions of CUDA, cuDNN, TensorRT, NCCL.

```
Do you wish to build TensorFlow with CUDA support? [y/N]: y
CUDA support will be enabled for TensorFlow.

Please specify the CUDA SDK version you want to use. [Leave empty to default to CUDA␣
↪9.0]: 10


Please specify the location where CUDA 10.0 toolkit is installed. Refer to README.md␣
↪for more details. [Default is /usr/local/cuda]:


Please specify the cuDNN version you want to use. [Leave empty to default to cuDNN 7]:


Please specify the location where cuDNN 7 library is installed. Refer to README.md␣
↪for more details. [Default is /usr/local/cuda]:


Do you wish to build TensorFlow with TensorRT support? [y/N]: y
TensorRT support will be enabled for TensorFlow.

Please specify the location where TensorRT is installed. [Default is /usr/lib/x86_64-
↪linux-gnu]:


Please specify the NCCL version you want to use. If NCCL 2.2 is not installed, then␣
↪you can use version 1.3 that can be fetched automatically but it may have worse␣
↪performance with multiple GPUs. [Default is 2.2]: 2.3
```

(continues on next page)

(continued from previous page)

```
Please specify the location where NCCL 2 library is installed. Refer to README.md for␣
↪more details. [Default is /usr/local/cuda]:
```

set environmental variable

```
export TF_NEED_TENSORRT=1
```

Edit "tensorflow/BUILD", add the following code to tf_cc_shared_object of the file.

```
"//tensorflow/contrib/tensorrt:trt_engine_op_kernel",
"//tensorflow/contrib/tensorrt:trt_engine_op_op_lib",
```

```
sed -i '/\"\/\/tensorflow\/cc:cc_ops\",/a\"\/\/tensorflow\/contrib\/tensorrt:trt_
↪engine_op_kernel\",\n\"\/\/tensorflow\/contrib\/tensorrt:trt_engine_op_op_lib\",'␣
↪tensorflow/BUILD
```

Build

```
bazel build --config=opt --config=cuda  //tensorflow:libtensorflow_cc.so \
--action_env="LD_LIBRARY_PATH=${LD_LIBRARY_PATH}"
```

1. Build deltann

```
cd delta/deltann && ./build.sh linux x86_64 tf
```

15.2 Tensorflow Lite

15.2.1 Build tensorflow lite for arm.

1. Config ndk Edit "tensorflow/WORKSPACE". Add the following code to the end of the file.

```
android_ndk_repository(
    name="androidndk",
    path="/ndk/path/android-ndk-r16b",
    api_level=21
)
```

1. Build tensoflow library

```
#armv7
bazel build -c opt --cxxopt=--std=c++11 \
    --config=android_arm //tensorflow/lite/experimental/c:libtensorflowlite_c.so

#arm64
bazel build -c opt --cxxopt=--std=c++11 \
    --config=android_arm64 //tensorflow/lite/experimental/c:libtensorflowlite_c.so
```

1. Build deltann

```
cd delta/deltann && ./build.sh android arm tflite
```

15.2.2 Build TensorFlow lite for iOS

1. You need to run a shell script to download the dependencies you need:

```
tensorflow/lite/tools/make/download_dependencies.sh
```

1. Build the library for all five supported architectures on iOS:

```
tensorflow/lite/tools/make/build_ios_universal_lib.sh
```

The resulting library is in tensorflow/lite/tools/make/gen/lib/libtensorflow-lite.a.

1. Build deltann

```
cd delta/deltann && ./build.sh ios arm tflite
```

15.2.3 Tailor tensorflow lite library

There isn't an automatic way of doing this. You can edit tensorflow/lite/kernels/register.cc and tensorflow/lite/kernels/BUILD, delete some ops that you don't require.

Egdelete lstm op if you don't require.

tensorflow/lite/kernels/register.cc

```
--- a/tensorflow/lite/kernels/register.cc
+++ b/tensorflow/lite/kernels/register.cc
@@ -60,7 +60,6 @@ TfLiteRegistration* Register_BATCH_TO_SPACE_ND();
 TfLiteRegistration* Register_MUL();
 TfLiteRegistration* Register_L2_NORMALIZATION();
 TfLiteRegistration* Register_LOCAL_RESPONSE_NORMALIZATION();
 -TfLiteRegistration* Register_LSTM();
 TfLiteRegistration* Register_BIDIRECTIONAL_SEQUENCE_LSTM();
 TfLiteRegistration* Register_UNIDIRECTIONAL_SEQUENCE_LSTM();
 TfLiteRegistration* Register_PAD();
@@ -184,7 +183,6 @@ BuiltinOpResolver::BuiltinOpResolver() {
   AddBuiltin(BuiltinOperator_L2_NORMALIZATION, Register_L2_NORMALIZATION());
   AddBuiltin(BuiltinOperator_LOCAL_RESPONSE_NORMALIZATION,
              Register_LOCAL_RESPONSE_NORMALIZATION());
-  AddBuiltin(BuiltinOperator_LSTM, Register_LSTM(), /* min_version */ 1,
               /* max_version */ 2);
   AddBuiltin(BuiltinOperator_BIDIRECTIONAL_SEQUENCE_LSTM,
   Register_BIDIRECTIONAL_SEQUENCE_LSTM());
```

tensorflow/lite/kernels/BUILD

```
--- a/tensorflow/lite/kernels/BUILD
+++ b/tensorflow/lite/kernels/BUILD
@@ -190,7 +190,6 @@ cc_library(
         "local_response_norm.cc",
         "logical.cc",
         "lsh_projection.cc",
-        "lstm.cc",
         "maximum_minimum.cc",
         "mfcc.cc",
         "mul.cc",
```

15.3 Tensorflow Seving

1. Download tensorflow serving

```
git clone https://github.com/tensorflow/serving.git
cd serving
```

1. Build tensorflow serving

```
bazel build //tensorflow_serving/model_servers:tensorflow_model_server
```

1. Build deltann

```
cd delta/deltann && ./build.sh linux x86_64 tfserving
```

15.4 Build in docker, using on bare metal

When link with `libx_ops.so`, `libdeltann.so` and `libtensorflow_cc.so`, `libtensorflow_framework.so`, mabe has problems as below:

```
/lib/deltann/lib/tensorflow/libtensorflow_cc.so: undefined reference to `std::_
↪V2::error_category::equivalent(std::error_code const&, int) const@GLIBCXX_3.4.21'
./lib/deltann/lib/tensorflow/libtensorflow_cc.so: undefined reference to `std::random_
↪device::_M_init(std::__cxx11::basic_string<char, std::char_traits<char>,␣
↪std::allocator<char> > const&)@GLIBCXX_3.4.21'
./lib/deltann/lib/deltann/libdeltann.so: undefined reference to `std::__cxx11::basic_
↪string<char, std::char_traits<char>, std::allocator<char> >::basic_string(char␣
↪const*, std::allocator<char> const&)@GLIBCXX_3.4.21'
./lib/deltann/lib/deltann/libdeltann.so: undefined reference to `std::__cxx11::basic_
↪string<char, std::char_traits<char>, std::allocator<char> >::_M_data()␣
↪const@GLIBCXX_3.4.21'
./lib/deltann/lib/deltann/libdeltann.so: undefined reference to `std::__cxx11::basic_
↪string<char, std::char_traits<char>, std::allocator<char> >::append(std::__
↪cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&
↪)@GLIBCXX_3.4.21'
./lib/deltann/lib/custom_ops/libx_ops.so: undefined reference to `std::out_of_
↪range::out_of_range(char const*)@GLIBCXX_3.4.21'
./lib/deltann/lib/custom_ops/libx_ops.so: undefined reference to `VTT for std::__
↪cxx11::basic_ostringstream<char, std::char_traits<char>, std::allocator<char> >
↪@GLIBCXX_3.4.21'
...
./lib/deltann/lib/custom_ops/libx_ops.so: undefined reference to `powf@GLIBC_2.27'
./lib/deltann/lib/tensorflow/libtensorflow_cc.so: undefined reference to `expf@GLIBC_
↪2.27'
./lib/deltann/lib/tensorflow/libtensorflow_cc.so: undefined reference to␣
↪`lgammaf@GLIBC_2.23'
./lib/deltann/lib/custom_ops/libx_ops.so: undefined reference to `logf@GLIBC_2.27'
./lib/deltann/lib/tensorflow/libtensorflow_cc.so: undefined reference to␣
↪`lgamma@GLIBC_2.23'
```

You need copy below librares from docker, and link with these. For glibc library are from https://www.gnu.org/software/libc.

```
    │   ├── glibc
    │   │   ├── ld-2.27.so
    │   │   ├── libc-2.27.so
    │   │   ├── libc.a
    │   │   ├── libc_nonshared.a
    │   │   ├── libc.so
    │   │   ├── libld-2.27.so
    │   │   ├── libm-2.27.so
    │   │   ├── libpthread-2.17.so
    │   │   ├── libpthread-2.27.so
    │   │   ├── libstdc++.so -> libstdc++.so.6
    │   │   ├── libstdc++.so.6 -> libstdc++.so.6.0.24
    │   │   └── libstdc++.so.6.0.24
```

```
DELTANN_DIR=./lib/deltann
DELTANNINC = $(DELTANN_DIR)/include
DELTANNLIB = -Wl,--start-group \
             -L$(DELTANN_DIR)/lib/custom_ops -lx_ops \
             -L$(DELTANN_DIR)/lib/deltann -ldeltann \
             -L$(DELTANN_DIR)/lib/tensorflow -ltensorflow_cc -ltensorflow_framework \
             -L$(DELTANN_DIR)/lib/glibc -lstdc++ -lm-2.27 -lld-2.27 -lpthread-2.27\
             -Wl,--end-group $(DELTANN_DIR)/lib/glibc/libc_nonshared.a
```

CHAPTER 16

Adding Tensorflow Op

All `custom-op` are under `delta/layers/ops/` directory.

16.1 Eigen Tensor

Eigen Tensor is unsupported eigen package, which is the underlying of Tensorflow Tensor.

16.2 Implement Op Kernel

Implement your op kernel class for underlying computing.

16.3 Create Tensorlow Op Wapper

Wapper the op kernel by Tensorflow Op or Tensorflow Lite Op.

16.4 Tensorflow

- Guide for New Op
- shape inference

16.5 Tensorflow-Lite

- TFLite custom ops.
- TFLite select ops

16.6 References

- custom-op
- lingvo
- Tensorflow Ops
- Tensorflow Lite Ops

CHAPTER 17

Serving

17.1 TF-Serving

17.1.1 Install

17.1.2 Developing with Docker

17.1.3 Pack your model into docker

17.1.4 Support Custom Ops

CHAPTER 18

TensorRT

The core of TensorRT™ is a C++ library that facilitates high performance inference on NVIDIA graphics processing units (GPUs). It is designed to work in a complementary fashion with training frameworks such as TensorFlow, Caffe, PyTorch, MXNet, etc. It focuses specifically on running an already trained network quickly and efficiently on a GPU for the purpose of generating a result (a process that is referred to in various places as scoring, detecting, regression, or inference).

18.1 Working With TensorFlow

TensorFlow integration with TensorRT(TF-TRT) optimizes and executes compatible subgraphs, allowing TensorFlow to execute the remaining graph. While you can still use TensorFlow's wide and flexible feature set, TensorRT will parse the model and apply optimizations to the portions of the graph wherever possible.

You will need to create a SavedModel (or frozen graph) out of a trained TensorFlow model, and give that to the Python API of TF-TRT, which then:

- returns the TensorRT optimized SavedModel (or frozen graph).
- replaces each supported subgraph with a TensorRT optimized node (called TRTEngineOp), producing a new TensorFlow graph.

During the TF-TRT optimization, TensorRT performs several important transformations and optimizations to the neural network graph. First, layers with unused output are eliminated to avoid unnecessary computation. Next, where possible, certain layers (such as convolution, bias, and ReLU) are fused to form a single layer. Another transformation is horizontal layer fusion, or layer aggregation, along with the required division of aggregated layers to their respective output. Horizontal layer fusion improves performance by combining layers that take the same source tensor and apply the same operations with similar parameters.

18.2 Reference

- TF-TRT
- TF-TRT User Guide

CHAPTER 19

Model Optimization

19.1 Quantization

- Quantization-aweare training
- Post-training quantization

19.2 Prouning

19.3 Compression

CHAPTER 20

Contributing Guide

20.1 License

The source file should contain a license header. See the existing files as the example.

20.2 Name style

All name in python and cpp using snake case style, except for `op` for `Tensorflow`. For Golang, using Camel-Case for `variable name` and `interface`.

20.3 Python style

Changes to Python code should conform the Chromium Python Style Guide.You can use yapf to check the style.The style configuration is `.style.yapf`.You can using `tools/format.sh` tool to format code.

20.4 C++ style

Changes to C++ code should conform to Google C++ Style Guide.You can use cpplint to check the style and use clang-format to format the code.The style configuration is `.clang-format`.You can using `tools/format.sh` tool to format code.

20.5 C++ macro

C++ macros should start with `DELTA_`, except for most common ones like `LOG` and `VLOG`.

20.6 Golang style

For Golang styple, please see docs below:

- How to Write Go Code
- Effective Go
- Go Code Review Comments
- Golang Style in Chinese

Before commit golang code, plase using `go fmt` and `go vec` to format and lint code.

20.7 Logging guideline

For `python` using abseil-py, more info.

For C++ using abseil-cpp, more info.

For Golang using glog.

20.8 Unit test

For `python` using `tf.test.TestCase`, and the entrypoint for python unittest is `tools/test/python_test.sh`.

For C++ using googletest, and the entrypoint for C++ unittest is `tools/test/cpp_test.sh`.

For Golang using `go test` for unittest.

CHAPTER 21

Released Models

21.1 NLP Models

21.1.1 Sequence classification

We provide the sequence classification models using CNN, LSTM, HAN (hierarchical attention networks), transformer, etc.

21.2 Sequence labeling

We provide the LSTM based sequence labeling and an LSTM with CRF based method.

21.3 Pairwise modeling

We implement the match of text pairwise models computing similarity across sentence representation encoded with two LSTM.

21.4 Sequence-to-sequence (seq2seq) modeling

We implement the standard seq2seq models using LSTM with attention and transformers. Note that, the seq2seq structure is also used for speech recognition. In DELTA, this part is shared between NLP and ASR tasks.

21.5 Multi-task modeling

We implement a multi-task model for sequence classification and labeling, where the sequence level loss and the step level loss are computed simultaneously. This model is used to jointly train an intent recognizer and named entity recognizer together.

21.6 Pretraining integration

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.

21.7 Speech models

21.7.1 Automatic speech recognition (ASR)

We provide an attention based seq2seq ASR model. We also implement another popular type of ASR model using connectionist temporal classification (CTC).

21.7.2 Speaker Verification/Identification

We provide an X-vector text-independent model and an end-to-end model.

21.7.3 Speech emotion recognition

Recently several deep learning based approaches have been successfully used in speech emotion recognition and we implement some models the in DELTA.

21.8 Multimodal models

21.8.1 Textual+acoustic

In our implementation, we use two sequential models (e.g.,CNNs or LSTMs) to learn the sequence embedding for speech and text separately, and thenconcatenates the learned embedding vectors for classification.

21.8.2 Textual+numeric

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.

CHAPTER 22

FAQ

22.1 Install

1. How to speed up the installation?

If you are a user from mainland China, you can use the comments code in `tools/install/install-delta.sh`.

```
# conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free/
# conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main/
# conda config --set show_channel_urls yes
# pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
```

1. `CondaValueError: prefix already exists: ../miniconda3/envs/delta-py3.6-tf2.0.0` or `ERROR: unknown command "config"`

Please update your conda by:

```
conda update -n base -c defaults conda
```

1. Custom operator error: `tensorflow.python.framework.errors_impl.NotFoundError: /../delta/delta/layers/ops/x_ops.so: undefined symbol: _ZN10tensorflow8str_util9LowercaseEN4absl11string_viewE`

This error always raise when you use the tensorflow installed by conda instead of pip. Conda use more high level gcc than pip dose to compile tensorflow. In this case, compilation of custom op with g++ 4.8 may cause this error.

You can use `conda install -c conda-forge cxx-compiler` to update the g++ version under your conda env.

then, compile custom op again

```
pushd delta/layers/ops/
./build.sh delta
popd
```

1. Segmentation fault. 0x00007fff48e930d4 in tensorflow::shape_inference::UnchangedShape(tensorflow::shape_inference::Inferenc
   ()

This error always raise when you use the tensorflow installed by pip instead of conda. The pip is compiled by g++ 4.8. In this case, you need to install g++ 4.8 on your system and re-compile your custom op again.

The error no.3 and no.4 are similar questions. The principle is to keep the g++ version for tensorflow compilation and custom compilation same. You need to upgrade or downgrade your g++ according to the cases.

CHAPTER 23

References

- Tensorflow
- lingvo
- Tensor2Tensor
- Kaldi
- ESPnet
- models
- YellowFin
- yapf
- python_speech_features
- abseil-cpp

CHAPTER 24

Version

Version No.

```
v{major}.{minor}.{stage}.{revision}
```

| stage | No. | description | e.g | | — | — | — | — | | Alpha | 0.5 | smoke test, estimate gains | v0.0.5.0 | | Beta | 0.7 | integration test | v0.0.7.2 | | RC1 | 0.8 | stress test | v0.0.8.1 | | RC2 | 0.9 | AB-test, evaluate gains | v0.0.9.0 | | Release | 1.0 | production | v0.1.0.0 |

CHAPTER 25

Release Version

Make sure all PRs under milestone `v0.3.2` are closed, then close the milestone. Using below command to generate relase note.

```
python tools/release_notes.py -c didi delta v0.3.2
```