Setup von ESPnet

Allgemein ist es empfehlenswert ESPnet2 zu verwenden, da für `ESPnet` eine kompilierte Version von Kaldi benötigt wird. Kaldi hat einige Dependencies zu anderen Bibliotheken, welches das Setup nocheinmal erschwert. Bei der Verwendung von `ESPnet2` wird jedoch trotzdem Kaldi benötigt, da die Hilfstools von Kaldi zur Vorverarbeitung der Daten verwendet werden. Hierzu muss Kaldi nicht kompiliert, aber in das `ESPnet`-Verzeichnis gelinked werden:

```
git clone https://github.com/espnet/espnet

git clone https://github.com/kaldi-asr/kaldi

cd espnet/tools

ln -s ../kaldi-asr
```

Verwendung der Google Colabs

Leider sind die Demos im offiziellen `ESPnet`-Repository outdated und nicht ausführbar.

Jedoch kann es sinnvoll sein zu Beginn ein Colab mit `ESPnet` aufzusetzen, wenn keine GPU auf dem eigenen Rechner vorhanden ist oder man flexibler arbeiten möchte, da Colab auf jedem System nutzbar ist (Windows ist als System für das Training eines Models nicht supported, der einzige Weg ist das Training über einen Docker Container).

Mac

Zuerst muss das Repository gecloned werden(falls noch nicht geschehen):

```
git clone https://github.com/espnet/espnet
```

Dann Installation der Pakete:

```
cd espnet

python3.7 -m pip install -e .
```

Es kann auch versucht werden, die Python Bibliothek direkt zu installieren beschrieben in der Installation:

```
pip install torch
pip install chainer==6.0.0 cupy==6.0.0    # [Option] If you'll use ESPnet1
pip install torchaudio                    # [Option] If you'll use
```

```
enhancement task
pip install torch_optimizer               # [Option] If you'll use
additional optimizers in ESPnet2
```

Es gibt verschiedene Abhängigkeiten zu den jeweiligen Paketen die nach und nach installiert werden müssen (werden in der Konsole ausgegeben, wenn sie nicht bereits installiert sind -> einfach mit augegebener Version per `pip install` installieren)

Daraufhin kann die Bibliothek installiert werden:

```
pip install espnet
```

Linux

Bei Verwendung von Linux können die gleichen Schritte wie bei Mac OS durchlaufen werden. Bei fehlen von Paketen können diese beim Tutorial oder beim später folgenden `Dockerfile` beim Abschnitt Ausführung eines pretrained Models gefunden werden.

Aufbau des ESPnet Projekts

Um sich im Projekt zurechtzufinden ist die Beschreibung der Directory-structure in der offiziellen `ESPnet`-Doku empfehlenswert.

Im Unterordner `egs2` sind die Beispiele für `ESPnet2` zu finden, nachdem man sich für ein Beispiel entschieden hat (die Beispiele sind nach Datensätzen benannt: Librispeech, Commonvoice usw.) kann man mit Hilfe des Tutorials ein Model trainieren.

Das `run.sh`-Skript ist dabei der zentrale Einstiegspunkt, um ein Model zu trainieren oder auszuführen (hier am Beispiel vom Librispeech-Skript):

```
#!/usr/bin/env bash
# Set bash to 'debug' mode, it will exit on :
# -e 'error', -u 'undefined variable', -o ... 'error in pipeline', -x
'print commands',
set -e
set -u
set -o pipefail

train_set="train_960"
valid_set="dev"
test_sets="test_clean test_other dev_clean dev_other"

asr_config=conf/tuning/train_asr_conformer7_n_fft512_hop_length256.yaml
lm_config=conf/tuning/train_lm_transformer2.yaml
inference_config=conf/decode_asr.yaml

./asr.sh \
```

```
    --lang en \
    --ngpu 16 \
    --nbpe 5000 \
    --max_wav_duration 30 \
    --speed_perturb_factors "0.9 1.0 1.1" \
    --asr_config "${asr_config}" \
    --lm_config "${lm_config}" \
    --inference_config "${inference_config}" \
    --train_set "${train_set}" \
    --valid_set "${valid_set}" \
    --test_sets "${test_sets}" \
    --lm_train_text "data/${train_set}/text data/local/other_text/text" \
    --bpe_train_text "data/${train_set}/text" "$@"
```

Im `run.sh`-Skript können die einzelnen `config.yaml` der ASR und Language Model so wie Pfade zu den Datasets(`--train_set`, `--valid_set`, `--test_set`) gesetzt werden. Aber auch die Sprache über `--lang` oder die Anzahl der GPUs(`--ngpu`) können gesetzt werden.

Dabei werden bestimmte Parameter an das `asr.sh`-Skript übergeben, welches alle möglichen Optionen zur Ausführung bietet.

Im `asr.sh`-Skript sind auch die einzelnen Stages zu erkennen:

Data Preparation Stages:

- Stage 1: Data preparation for data
- Stage 2: Speed perturbation
- Stage 3: Feature extraction
- Stage 4: Remove long/short data
- Stage 5: Generate token_list from bpe_train_text using bpe/ Generate character level token_list from lm_train_text

Language Model related Stages:

- Stage 6: LM collect stats
- Stage 7: LM Training
- Stage 8: Calc perplexity

ASR related Stages:

- Stage 9: ASR collect stats
- Stage 10: ASR Training

Model Evaluation:

- Stage 11: Decoding
- Stage 12: Scoring

Uploading and Packing of Model:

- Stage 13: Pack model
- Stage 14: Upload model to Zenodo (ESPnet ModelZoo)

Es ist also möglich nur Teile des Models zu trainieren, indem man bei der Ausführung den `--stage` und den `--stop_stage`-Parameter setzt:

```
./run.sh --stage 6 --stop_stage 8
```

Beim gezeigten Beispiel würde man nur die Language Model Stages durchlaufen.

Ausführung eines pretrained Models

Die bereits vorhandenen Modelle sind unter dem Projekt ESPnet Model Zoo zu finden. Dort gibt es ein einfaches Tutorial unter ASR zum Ausführen eines einzelnen Models. Die bereits trainierten Modelle sind in folgender Tabelle aufgelistet. Der einfachste Weg, um ein pretrained Model zu verwenden ist das Erstellen eines Docker Containers mit Hilfe eines `Dockerfiles`:

```
FROM ubuntu:18.04

ARG DOCKER_VER
ENV DOCKER_BUILT_VER ${DOCKER_VER}}

ARG NUM_BUILD_CORES=8
ENV NUM_BUILD_CORES ${NUM_BUILD_CORES}

COPY /test_usage /

RUN DEBIAN_FRONTEND=noninteractive apt-get update && apt-get -y install --
no-install-recommends \
        software-properties-common \
        automake \
        autoconf \
        apt-utils \
        bc \
        build-essential \
        ca-certificates \
        cmake \
        curl \
        flac \
        ffmpeg \
        gawk \
        gfortran \
        git \
        libtool \
        libsndfile1-dev \
        python2.7 \
        python3-pip \
        python3.7 \
```

```
        python3-setuptools \
        python3-wheel \
        sox \
        subversion \
        unzip \
        wget \
        zip \
        zlib1g-dev \
        llvm-9 \
        && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

RUN export LLVM_CONFIG=/usr/bin/llvm-config-9

RUN apt-get update

RUN apt install -y python3.7-dev

RUN python3.7 -m pip install -r requirements.txt

RUN git clone https://github.com/espnet/espnet
RUN git clone https://github.com/kaldi-asr/kaldi

RUN cd espnet/tools \
    ln -s ../kaldi-asr

RUN cd espnet \
python3.7 -m pip install -e .

WORKDIR /
```

Im Container werden alle wichtigen Pakete installiert und alle nötigen Dateien aus dem `test_usage` Unterverzeichnis in den Container kopiert. Im gleichnamigen file `test_usage.py` kann ein Model an einem `.wav`-file getestet werden:

```
speech2text = Speech2Text(
    **d.download_and_unpack("model_name"),
    # Decoding parameters are not included in the model file
    maxlenratio=0.0,
    minlenratio=0.0,
    beam_size=20,
    ctc_weight=0.3,
    lm_weight=0.5,
    penalty=0.0,
    nbest=1
)
```

`model_name` ist mit dem Name des gewünschten Models aus der Tabelle zu ersetzen. Zusätzlich muss die zu testende Audio-Datei in den `test_usage` Ordner hinzugefügt und in der `test_usage.py` die Zeile mit

dem Dateinamen angepasst werden:

```
speech, rate = soundfile.read("speech.wav")
```

Im Ordner mit dem Dockerfile muss der Container gebaut werden:

```
docker build . -t <imagename>
```

Daraufhin muss der angegebene `imagename` genutzt werden, um einen interaktiven Container zu starten:

```
docker run -it <imagename> /bin/bash
```

Dann kann man im Container einfach die `test_usage.py` ausführen und überprüfen, was das Model für eine Transkription für die zuvor angegebene `.wav` ausgibt.

Finetuning eines Models(Transfer learning)

Beim Finetuning eines Models wird ein pretrained Model(generalisiert trainiertes Model) verwendet und auf den jeweiligen Anwendungsfall spezialisiert. Dabei gilt es zu beachten, dass hierzu zwei Methoden angewandt werden können:

- Freezen der Parameter des pretrained Models
- Retraining der ganzen Gewichte (kein freezing)

Zusätzlich sollte man überlegen, welche Teile des Systems man neu trainieren möchte:

- ASR-Model
- Language Model

Im Language Model sind die Wahrscheinlichkeiten der aufeinanderfolgenden Wörter enthalten (welche mit Text aus Rohdaten trainiert werden). Dieses Language Model wird genutzt, um schlüssigere Ergebnisse zu erzielen.

Nur mit Textdaten

Wenn man nur Textdaten zur Verfügung hat, um sein System zu finetunen empfiehlt es sich das Language Model zu trainieren (da dieses auf reinen Textdaten basiert).

Dafür müssen in der Ordnerstruktur folgende Dateien mit den neuen Daten enstehen:

```
├── data
│   └── token_list
│       └── bpe_unigram5000
│           ├── bpe.model
```

```
│           └── tokens.txt
├── dump
│   └── raw
│       ├── dev
│       │   └── text
│       ├── srctexts
│       └── test_clean
│           └── text
```

Bei dem Ordner `dump` handelt sich um den Ordner für die neuen Trainingsdaten. Diese sollten aufgeteilt werden (Aufteilung darf variieren):

- `train` (75% der Daten)
- `validation` (15%)
- `test`(10%)

Damit die Files zum Training oder zur Validierung verwendet werden können, gilt es entweder die Benennung der Files an die `config.yaml` des pretrained models anzupassen oder die Pfade in der config selbst anzupassen (sonst werden sie nicht gefunden). Festgelegt werden die Pfade an folgender Stelle in der config:

```
train_data_path_and_name_and_type:
-   - dump/raw/srctexts
    - text
    - text
valid_data_path_and_name_and_type:
-   - dump/raw/dev/text
    - text
    - text
```

Der Aufbau des `srctexts`, welches alle Trainingsdaten enthält, ist Kaldi nachempfunden und ist nach folgendem Format zu erstellen:

```
ut_1 why should i use rasa instead of google dialogflow?
ut_2 why should I use rasa?
ut_3 why should I switch to rasa?
```

Dabei ist das Tutorial von Kaldi hilfreich, um zu verstehen, welche Daten selbst erstellt werden müssen und welche durch die `Data-Preparation`-Skripts generiert werden.

Beim trainieren muss die Größe der sogenannten `token_list` unverändert sein, da sonst das Training nicht gelingt. Diese `token_list` enthält alle aus den Daten bekannten `tokens`.

Ist keine `token_list` im pretrained Model vorhanden, wurde ein sogenanntes `bpe_model` verwendet, welches eine andere Möglichkeit darstellt, um alle trainierten Tokens zu speichern.

```
exp
├── asr_stats_raw_sp
│   └── train
│       └── feats_stats.npz
├── asr_train_asr_transformer_e18_raw_bpe_sp
│   ├── 54epoch.pth
│   ├── RESULTS.md
│   └── config.yaml
└── lm_train_lm_adam_bpe
    ├── 20epoch.pth
    ├── checkpoint.pth
    ├── config.yaml
    ├── data
    │   └── token_list
    │       └── bpe_unigram5000
    │           ├── bpe.model
    │           └── tokens.txt
    ├── dump
    │   └── raw
    │       ├── dev
    │       │   └── text
    │       ├── srctexts
    │       └── test_clean
    │           └── text
    │
```

Trainiert werden kann das Model dann durch folgenden Befehl:

```
python3 -m espnet2.bin.lm_train --config config.yaml --ngpu 0 --
dist_world_size none
```

Es müssen in der `config.yaml` die folgenden Zeilen nach jeder Ausführung gelöscht werden:

```
required:
- output_dir
- token_list
distributed: false
```

Zusätzlich sollte in der config der Name des Models(welches trainiert werden soll) unter folgendem Parameter gestetzt werden:

```
init_param:
- 20epoch.pth
```

Es wird `python` oder `python3` zur Ausführung verwendet je nach dem, wo alle Pakete im Setup installiert wurden.

Durch die Option `--ngpu 0` kann das Training auf die CPU umgestellt werden, falls keine GPU zur Verfügung steht.

Durch `--dist_world_size none` wird das distributed training deaktiviert (da dies beim CPU training nicht benötigt wird).

Verwendung eines eigenen Models

Hierzu kann das gleiche `test_usage.py`(entspricht dem ASR-Beispiel von ESPnet ModelZoo) genutzt werden wie im Docker-Container. Dabei gilt es nur das Skript entsprechend anzupassen, damit nicht ein externes Model heruntergeladen wird. Dazu muss folgende Zeile mit den Pfaden zum lokal trainierten Model und dessen `config.yaml` angepasst werden:

```
speech2text = Speech2Text("config.yaml", "custom_model.pth",
maxlenratio=0.0, minlenratio=0.0,
beam_size=1,ctc_weight=0.3,lm_weight=0.5,penalty=0.0,nbest=1)
```

Verwendung eines eigenen Language Models

Damit ein separates language model im `test_usage.py`-Skript verwendet werden kann, müssen die notwendigen Pfade an das `asr_inference.py`-Skript übergeben werden. Die einfachste Variante zum Testen, ist das eintragen der Pfade direkt in der Datei (zu finden unter `espnet/espnet2/bin/asr_inference.py`) :

```
 def __init__(
        self,
        asr_train_config: Union[Path, str],
        asr_model_file: Union[Path, str] = None,
        lm_train_config: Union[Path, str] =
"lm_train_lm_adam_bpe/config.yaml",
        lm_file: Union[Path, str] = "lm_train_lm_adam_bpe/20epoch.pth",
        token_type: str = None,
        bpemodel: str = None,
        device: str = "cpu",
        maxlenratio: float = 0.0,
        minlenratio: float = 0.0,
        batch_size: int = 1,
        dtype: str = "float32",
        beam_size: int = 20,
        ctc_weight: float = 0.5,
        lm_weight: float = 1.0,
        penalty: float = 0.0,
        nbest: int = 1,
    ):
```

Hilfreiche Links

- ESPnet2-Dokumentation
- Espnet2 Training Configuration
- ESPnet Projekt
- ESPnet Installation
- ESPnet ModelZoo
- Kaldi Data Preparation
- Kaldi Data Preparation - Audio/Acoustic/Language Data

Häufige Probleme

- No module named setuptools
- ESPnet Github Issues

Hilfreiche Issues zum Language Model training:

- Domain adaptation for existing model #2845
- how to retrain just with LM model in asr.sh #2992
- Fine-tuning a pre-trained ASR model on a new dataset #2930

Mögliche fehlende Pakete:

```
librosa (0.8.0)
numba==0.50.1
torch==1.7.1
numpy==1.19.4
SoundFile==0.10.3.post1
espnet-model-zoo==0.0.0a20
scipy==1.5.4
Cython==0.29.21
```