KTH Computer Science
and Communication

DT2118 Lab3B: Deep Learning for ASR

1 Objective

The objective is build and train a Deep Neural Network for phonetic recognition. Optionally, you can build a full DNN-HMM model and run isolated word or continuous speech recognition using the functions you implemented in Lab 2. The focus is on analysing the effect of different training parameters on performance, and test different feature extraction methods. Most of the more technical steps that are not directly related to the learning objectives, for example managing file formats and preparing the data, are given by lab package.

In the process you should also familiarise with the following tools:

- The Hidden Markov Model Toolkit (HTK)[1] for training Hidden Markov Models and Gaussian Mixture Models (GMM-HMMs)
- the PDNN toolkit[2], which provides wrappers to Theano[3] specifically made for Speech Recognition.
- facilities at the Parallel Data Centre (PDC) at KTH[4]

2 Task

Train and test a phonetic recogniser based on digit speech material from the TIDIGIT database. The first part of the task is described in the instructions for Lab 3A, see Section 3 of these instructions for more details.

- use the provided scripts and an already trained GMM-HMM system to create time aligned phonetic transcriptions of the data
- train and test DNN models with different parameters
- compute state posteriors with the DNN
- perform frame-by-frame phoneme recognition
- compare the frame by frame results with those obtained with the GMMs

Optional:

- define a word level HMM based on the phonetic models in the GMM-HMM

[1] http://htk.eng.cam.ac.uk/

[2] https://www.cs.cmu.edu/~ymiao/pdnntk.html

[3] http://deeplearning.net/software/theano/

[4] https://www.pdc.kth.se/, this is also intended as an introduction to the work in the final project.

- perform isolated word recognition with the forward algorithm you implemented in Lab 2 and the DNN posteriors
- perform continuous speech recognition with the Viterbi decoder you implemented in Lab 2 and the DNN posteriors
- compare the results with the GMM-HMM results

In order to pass the lab, you will need to follow the steps described in this document, and produce a report where you describe your work and answer the questions asked here. The report should be submitted in the Assignment section in the course Web on KTH Social `https://www.kth.se/social/course/DT2118/`. One submission should be done for each group, clearly stating all the members of that group.

3 Prerequisite

In order to perform this lab exercise you need to download and carry out part of Lab 3A. Refer to the instructions for that lab for more information.

The steps that you need to complete are:

1. Section 3,
2. Section 4.1,
3. Section 4.2, only with `MFCC_0_D_A` features
4. Section 4.3

The rest of these instructions assume that you are in the root directory for Lab 3A, and have a `models_MFCC_0_D_A` subdirectory with the trained GMM-HMM models. It also assumes you have unpacked the files for the current lab package in the same directory.

Most of the lab can be performed on a CSC Ubuntu machine. The Deep Neural Network training is best performed on a GPU, for example by logging into `tegner.pdc.kth.se`.

In order to add all the required tools in the PATH if you are on a CSC Ubuntu machine, run `source tools/modules_csc`. If you are on `tegner.pdc.kth.se`, refer to Appendix A.

4 Data Preparation

To train a neural network we need more information than for Baum-Welch training:

1. we need to split the training data into training and validation sets for the stopping criterion,
2. we need frame level transcriptions of the HMM states
3. the features need to be normalised

Finally we need to explicitly compute the features (which in the previous lab were computed on the fly by HTK from wave files).

4.1 Training and Validation Sets

Divide the list of training files in `workdir/train.lst` into a training set (roughly 90%) and validation set (remaining 10%). Call the resulting files:

```
workdir/train_tr.lst
workdir/train_va.lst
```

Make sure that there is a similar distribution of men and women in both sets, and that each speaker is only included in one of the two sets. The last requirement is to ensure that we do not get artificially good results on the validation set. Explain how you selected the two data sets.

4.2 Forced Aligned Transcriptions (for Viterbi Training)

Create state level time aligned transcriptions for the training, validation and test sets using the GMM-HMM models you trained in the previous lab:

```
tools/forced_align_states.sh MFCC_0_D_A
```

Check the resulting files:

```
workdir/train_tr_align.mlf
workdir/train_va_align.mlf
workdir/test_align.mlf
```

Note that, compared to the `forced_align.sh` script used in Lab 3A, the `forced_align_states.sh` script adds the `-f` option in `HVite` that forces the program to output state-level time aligned transcriptions.

4.3 Feature Extraction and File Formats

In Machine learning problems that do not involve sequences, the feature files are simply $N \times M$ arrays with N examples of dimension M, where N is the number of examples in the full training, validation or test sets. In speech, as in any other sequential problems, we need to keep track of the sequence boundaries. There are a number of ways to do this:

- use a different file for each utterance
- use a list of arrays with one array per utterance (see Lab 1 and 2)
- use a single array, but store the length of each utterance in an additional array
- use specially designed data formats

The PDNN toolkit that you will use in the rest of the exercise supports two specially designed data formats: `pfile` and `kaldi`. We decided to use pfiles in this exercise because many examples in the documentation of PDNN use this format. For more information on pfiles check out `http://www1.icsi.berkeley.edu/Speech/faq/ftrformats.html`.

First we need to assign numerical ids to each HMM state that we want to model with the DNN (target states). The script `tools/phones2stateid.py` takes as input a list of phones and generates a list of states with incremental ids:

```
tools/phones2stateid.py workdir/phones1.lst > workdir/state2id.lst
```

Refer to the `workdir/state2id.lst` file when you want to convert the outputs of your DNN back to posteriors for the states, or if you want to merge states that belong to the same phoneme when you compute phoneme error rates.

Now compute the feature files. You can use any of the features listed in Lab 3A, although the rest of the lab suggest you compare `MFCC_0_D_A` and `FBANK` features. The script `tools/htk2kaldi.py` parses a Master Label File in HTK format, generates features and label files for each utterance and stores them in three files in Kaldi format.

For example:

```
tools/htk2pfile.py workdir/train_tr_align.mlf workdir/state2id.lst FBANK \
     workdir/train_tr_FBANK.pfile
tools/htk2pfile.py workdir/train_va_align.mlf workdir/state2id.lst FBANK \
     workdir/train_va_FBANK.pfile
tools/htk2pfile.py workdir/test_align.mlf workdir/state2id.lst FBANK \
     workdir/test_FBANK.pfile
```

4.4 Feature Normalisation

Normalise the features so that each feature coefficient has zero mean and unit variance. For a simple way of doing this over a whole set of observations in each file, check the `pfile_norm` command from `pfile_utils`. Discuss why normalising the validation and test sets this way is not entirely correct. As an alternative, you can normalise each utterance independently with `pfile_normutts`. What are the advantages and disadvantages of this strategy? If you wish to implement your own normalisation strategy, use the functions `pfile_read` and `pfile_write` in `tools/pfile.py` to read the data into `numpy` arrays, and to write it back to file after modification. These functions are also useful to read the labels to do the frame-by-frame evaluation required by the next steps.

5 Phoneme Recognition with Deep Neural Networks

Consult the documentation at the PDNN site: `https://www.cs.cmu.edu/~ymiao/pdnntk.html`.

With the help of PDNN, train the following networks. In each case, report the evolution of the training and evaluation errors as well as the learning rate (remember that errors are computed on a frame-by-frame basis and on states which are more detailed classes than phonemes):[5]

1. Deep Neural Network based on normalised `FBANK` features with 4 hidden layers of 1024 units each and rectified linear unit activation functions.
2. same as before but with a context input window of 5 frames in the past and in the future
3. same as at the previous point, but with only 256 units per hidden layer
4. same as at the first point, but based on normalised `MFCC_0_D_A` features instead of normalised `FBANK`

Choose one of the above networks for further evaluation (you can also evaluate all the networks if you want).

[5] The parameters for each training have been chosen in a way that the training would converge in less than 20 minutes using the default strategy for learning rate update implemented in PDNN.

- With the help of PDNN excite your favourite network with the test data and observe the resulting state posteriors (note that PDNN will save the result in a Pickle file)
- compute the Maximum a Posteriori class on a frame-by-frame basis and compare it to the reference transcription (error rate, confusion matrix, ...)
- compute the errors on a phoneme basis, that is, after merging states into the corresponding phoneme according to `workdir/state2id.lst`

Practical advice on PDNN:

- create a separate directory for each network you train and run the training commands from that directory (PDNN creates temporary files in the working directory that might interfere if you train more than one network at the same time)
- if you stop the training because you want to change something in the network, remove the network file before you start a new training, otherwise PDNN will read it and continue the training from where it stopped (this is of course a great feature if the training was stopped by external causes and you want to continue without loosing the previous epochs)
- although not required during training, use the option `--cfg-output-file`: The config file will be required when you excite the network with new data.

There are many other parameters that you can vary, if you have time to play with the models. For example:

- different activation functions than ReLU (but be aware that convergence is much slower)
- different number of hidden layers
- different number of nodes per layer
- different length of context input window
- strategy to update learning rate and momentum
- initialisation with DBNs instead of random

6 Optional: Test a Hybrid HMM-DNN System

This part will probably require more time than you have to complete the lab and is therefore optional. The reason why I included it, is to show how you can use the functions you implemented in the previous labs, to build a simple, but complete DNN-HMM recogniser.

6.1 Isolated Word Recognition

1. In the list of test files `workdir/test.lst`, select only the isolated digit files (the ones with name in the form `[oz1-9][ab].wav`.
2. Compute state posteriors for each utterance with your DNN.

3. estimate the a-priori probability of each state from the `workdir/train_tr_FBANK.pfile` file (you can use the function `pfile_read` to import the labels into Python)

4. convert posteriors to *scaled likelihoods* for the states using the a-priori probabilities at the previous points

5. reload the transition matrices in `models[i]['hmm']['transmat']` from Lab 2, with index from 0 to 10 for the 11 digits. Refer to `models[i]['digit']` to know which digit the model corresponds to.

6. Create a mapping between the states in the DNNs output layer and the indices in your word level transition matrices. To do this, refer to the pronunciation of each digit that you can find in `models[i]['pron']` and to the `workdir/state2id.lst` file.

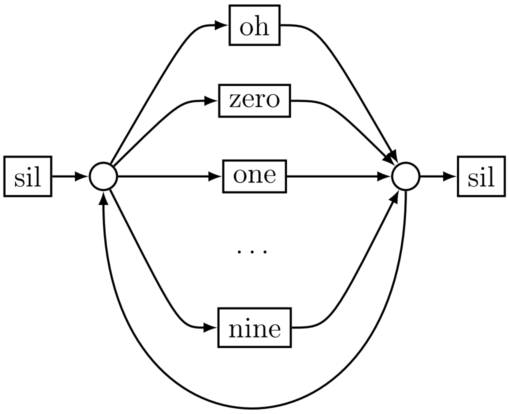
7. using your `forward` function from Lab 2, score each utterance with each transition model and the scaled likelihoods from the DNN (be careful to match the states ids in your word level models to those from the DNN output layer, see also a previous point)

8. classify each utterance according to the model with maximum likelihood.

9. compare your results with those obtained with the GMM-HMM.

6.2 Continuous Speech Recognition

1. combine the word level transition models from the previous section to form a transition matrix that describes the following model (notice that you need to remove the three silence states at the beginning and end of each `models[i]['hmm'][transmat]` before you combine them in the loop):

   Use a *flat* language model, that is, at every word transition, it is equally likely to access any of the 11 words.

2. excite the DNN with all the test data in `workdir/test.lst`, and get posterior probabilities for each utterance, each frame and each state in the model

3. using the Viterbi decoder you implemented in Lab 2, find the optimal word sequence in each utterance (be careful to match the state ids in your HMM model to those from the DNN output layer)

4. score the transcribed utterances using work accuracy or word error rate.
5. compare with the results obtained with the GMM-HMM.

A PDC Specific Instructions

In order to run the PDNN commands on GPUs, you need to login on `tengren.pdc.kth.se`. You can refer to the presentation from PDC you can find in the course web page for detailed information. However, the program `sbatch` to get time allocation and run programs in the background does not seem to work with Theano, so you will have to run `salloc` instead. Here is short guide:

1. First you need to authenticate with the help of kerberos on your local machine. From a machine where kerberos is installed and configure run:

   ```
   kinit -f -l 7d <username>@NADA.KTH.SE
   ```

   to get a 7 days forwardable ticket on your local machine. If you are using a CSC Ubuntu machine, run instead

   ```
   pdc-kinit -f -l 7d <username>@NADA.KTH.SE
   ```

   this will keep also the ticket `<username>@KTH.SE` allowing you to see the files in your home directory on AFS.

2. then you login with `ssh`, (or `pdc-ssh` on CSC Ubuntu)[6]:

   ```
   [pdc-]ssh -Y <username>@tegner.pdc.kth.se
   ```

3. on `tegner` you get time allocation running[7]:

   ```
   salloc -t <hours>:<minutes>:<seconds> -A edu16.DT2118 --gres=gpu:K80:2
   ```

4. you will get a message like the following:

   ```
   salloc: Granted job allocation 41999
   salloc: Waiting for resource configuration
   salloc: Nodes t02n29 are ready for job
   ```

   where the job number (41999) and the associated node (`t02n29`) will vary.

5. **From another teminal window on your local machine**, login on that specific node:

   ```
   ssh -Y t02n29.pdc.kth.se
   ```

   running `ssh` from `tegner.pdc.kth.se` to the node **will not work**

6. create a theano configuration file called `.theanorc` with the following content:

[6] see `https://www.pdc.kth.se/resources/software/login-1/.ssh-conf` to configure ssh correctly for PDC

[7] The last optional argument ensures you get a node with Nvidia Tesla K80 GPU , otherwise you will get a node with Nvidia Quadro K420. Refer to the page `https://www.pdc.kth.se/resources/computers/tegner/hardware` for more information about the hardware on tegner.

```
[global]
device = gpu0
floatX = float32
[nvcc]
fastmath = True
```

7. the lab requires several hundreds of MB of space. If you do not have enough space in your home directory, put the lab files under

   ```
   /cfs/klemming/nobackup/<first_letter_in_user_name>/<user_name>/
   ```

   remember that the data stored there is not backed up. If you need to copy the files back and forward with your local machine, check the `rsync` command

8. run the `screen` command. This will start a screen terminal, that will allow you to *detach* the terminal and logout without stopping the process you want to run[8]

9. in order to get the required software, from the lab main directory run

   ```
   source tools/modules_tegner
   ```

10. if everything went well, now you can run the PDNN scripts with, for example

    ```
    python $PDNNDIR/cmds/run_DNN.py ... |& tee -a logfile
    ```

    where the `tee` command will display the standard output and standard error of the training command in the terminal as well as appending it to the `logfile`

11. if you want to logout while the program is running, hit `ctrl+a` and then `d` to detach the screen and logout. When you login again into that node, you can run `screen -r` to reattach the terminal.

12. while you are logged in on the specific node, you can check CPU usage with the command `top` and GPU useage with the command `nvidia-smi`.

Use `squeue [-u <username>]` to see your time allocated jobs, and `scancel jobid` to remove a submitted job.

[8] Refer to the screen manual for further info http://linux.die.net/man/1/screen

B Required Software on Own Computer

If you perform the lab in one of the CSC Ubuntu computers, or on `tengren.pdc.kth.se`, all the required software is already installed and can be made available by running:

```
source tools/modules
```

or

```
source tools/modules_tegner
```

from the lab package main directory.

If you wish to perform the lab on your own computer, you will need to install the required software by hand. Here is a short guide based on Ubuntu 14.04, if you encounter problems, the advice is to run on CSC computers instead.

B.1 HTK

1. Download stable package (3.4.1) from `http://htk.eng.cam.ac.uk/` (this requires registering to the site)
2. extract the content with your favourite archive tool, for example from the command line:

   ```
   cd Downloads
   tar xvfz HTK-3.4.1.tar.gz
   cd htk
   ```

3. Install the required dependencies:

   ```
   sudo apt-get install libc6-dev-i386
   ```

4. configure and build HTK (you will get many warnings, but no error, hopefully):

   ```
   ./configure --disable-hslab --prefix=/opt/htk
   make
   ```

5. install the software in the corresponding directory and add the path to your PATH:

   ```
   sudo mkdir /opt/htk
   sudo make install
   echo "export PATH=/opt/htk/bin:\$PATH" >> ~/.bashrc
   ```

6. test the installation: open a new terminal and run `HList`. You should get a usage description for the command

B.2 Wavesurfer

This can be useful to visualise the results (label files) together with the wave files. Wavesurfer is part of the apt-get repositories:

```
sudo apt-get install tk8.5 libsnack-alsa wavesurfer
```

B.3 PFile Utils

Needed by the `tools/pfile.py` and `tools/htk2pfile.py` scripts to generate pfiles for PDNN:

```
wget ftp://ftp.icsi.berkeley.edu/pub/real/davidj/quicknet.tar.gz
tar -xvzf quicknet.tar.gz
cd quicknet-v3_33/
./configure --prefix=`pwd`
make install
cd ..
wget http://www.icsi.berkeley.edu/ftp/pub/real/davidj/pfile_utils-v0_51.tar.gz
tar -xvzf pfile_utils-v0_51.tar.gz
cd pfile_utils-v0_51
./configure --prefix=`pwd` --with-quicknet=`pwd`/../quicknet-v3_33/lib
make -j 4
make install
```

B.4 Theano

Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. It is the building block on which PDNN is created. Follow the instructions here: `http://deeplearning.net/software/theano/install_ubuntu.html`

B.5 PDNN

PDNN is a set of scripts that use Theano to create, train and test various kinds of Deep Neural Networks. To install run:

```
cd location_of_choice
git clone https://github.com/yajiemiao/pdnn
export PYTHONPATH=`pwd`/pdnn:$PYTHONPATH
```