Neural FX lab: Training neural networks

In this lab you are going to experiment (in Python) with some neural networks that process audio. The aim is to see how the networks perform at modelling some basic signals and systems.

Setup your torch python environment

If you are familiar with python and virtual environments/ anaconda, you should simply create a new virtual environment and pip install torch, numpy and soundfile. If you need instructions on how to do this, in a shell/ terminal:

Macos/ linux:

```
cd to the folder you want to work in
python3 -m venv neural_fx_env
source neural_fx_env/bin/activate
pip install torch numpy soundfile
```

Windows (assuming you installed python)

```
py -m venv neural_fx_env
neural_fx_env/Scripts/activate
pip install torch numpy soundfile
```

Verify your setup

Download the zip containing the code for this lab. There should be a python folder.

```
# assuming you are in your activated virtualenv
cd 035c_train_lstm/python
python train.py
```

If it does not complain about missing modules, you are good to go.

Prepare some data

We want to see how well a simple LSTM with a simple training method can model some different signals. The first step is to generate some processed audio using known types of effects.

You have been provided with an example of training data in the data folder:

```
.
|-- audio_audacity_dist
|   |-- input
|   |   `-- clean_signal.wav
|   `-- output
|       `-- audacity_dist.wav
```

To create your own example, load clean_signal.wav into audacity, process it using some sort of filter or other effect, then save it as a wav file, @44100Hz sample rate. It only reads the first channel of mono files.

Then, set the folder structure up as above:

- Create a top level folder called 'my_processed_audio'
- Create subfolders: input and output
- Put your processed file in output and your clean file in input

Train a model

Now edit the file train.py so it points at your data folder. Edit this variable in train.py:

```
audio_folder = "../../data/audio_audacity_dist"
```

Run the training script

```
python train.py
```

Every 50 epochs, the script will save out an example of how the network processed audio. It will load in 'guitar.wav' from the top level data folder and render it to your data sub folder, e.g. audacity_dist0.wav is the result at epoch zero.

As the training progresses, you can listen to the results.

Experiment!

Now it is time to experiment. Here are some things you can try:

- Try processing the audio using various filters: high pass, low pass, resonant. Can the network learn these filters?
- Try more or less hidden units on the LSTM
- Do you have any non-linear effects? Try those
- More advanced: try some other network layers such as conv1D.

You might want to work on the training script to more systematically run these experiments. You can also calculate the loss on the test set (test_dl) as a metric for success.

Loading the model into C++

The models can be loaded into C++ using RTNeural. You will see in your python folder that a 'model.json' file is saved every time a new record validation score is set. This file contains the weights for your model in JSON format.

If you want to experiment with this, check out the main.cpp file in the src folder. When you load the model into C++, you need to specify the architecture as this is not saved in model.json.

This line in main.cpp specifies the model architecture:

```
using ModelType = RTNeural::ModelT<float, 1, 1, RTNeural::LSTMLayerT<float, 1, 8>, RTNeural
```

In that case, one lstm layer with one input and 8 hidden states and one fully connected (linear/ dense) layer with 8 inputs and 1 output.

The model weights are then individually loaded into the LSTM model in these lines:

```
// model is type 'ModelType' as above
auto& lstm = model.get<0>();
// load the first layer from a json object
RTNeural::torch_helpers::loadLSTM<float> (modelJson, "lstm.", lstm);
// load the second layer
RTNeural::torch_helpers::loadDense<float> (modelJson, "dense.", dense);
```

Importantly, the layers in the model.json file must have the lstm. and dense. prefixes for the parser to find the weights. In mt model.json, I see a key: "lstm.weight_ih_l0".

Use the regular cmake commands to build and run the program:

```
cmake -B build .
cd build
cmake --build .
./rtneural-lstm
```