Training a LSTM RNN Classifier Using Neuromorphic Data (AEDAT)

Operations Summary

In order to achieve training, prepare data as described in the following 'DataArchitecture Base Requirements' section using external preparation tools as described in the 'External Preparation Tools' section, if you wish to streamline the process. Once data is prepared, set desired training parameters following guidelines and documentation presented in the 'Base Training Parameters' section. You may, after this,run training as described in the 'Execution' section making sure you follow specific guidelines adequate in desired case.

Python Dependencies

The following python libraries are required by the software:

```
Keras>=2.0.2
numpy>=1.12.1
pandas>=0.19.2
tqdm>=4.11.2
matplotlib>=2.0.0
Pillow>=2.1.0
h5py>=2.7.0
ffmpeg-python>=0.2.0
```

Data Architecture Base Requirements

The first step is to generate videos using AEDAT files as a source. This can be achieved using jAER in conjunction with a Python script built to handle the task automatically. You can install jAER by running the 'install_jaer.sh' script. Once done, use the 'run_avi.py' script to generate videos. You can specify the number of events per frame you want to use here. Place generated AVI videos into the data folder, within adequate 'train' and 'test' subfolders, within class folders as define earlier.

Next, prepare your lists. Locate the 'list' folder within the 'data' folder. Within said folder, create a 'classInd.txt' text file. This file will hold class indexes; each class should have a unique integer identifier and a one word name, for example 'Pencil'. As a convention, we capitalize the first letter of each class. The below example is given for four classes.

```
1 Burst
2 Sine
3 Square
4 Triangle
```

The next step is to organize data in classes for both training and testing sets. Inside the 'data' folder, create a 'train' folder and a 'test' folder. Put your classes inside the 'train' and 'test' folders. You should have training and testing data for all classes in your set with identically named folders, but containing different data. If the class name is present in the filename of each file, this process can be automated by the '1_move_files.py' script.

Once done, create a 'testlist01.txt' text file. This file will hold relative paths to test AVI videos. It can be generated using external preparation scripts, which is especially convenient when handling large amounts of examples. The final task is to create a 'trainlist01.txt' text file that holds relative paths to train AVI videos, as well as associated class index, for each file. The below example is a valid 'trainlist01.txt' example line.

```
Sine/Opt7V-2Hz-Sine.avi 2
```

As one can see, the path and the class index are separated by a simple space. As it is the case with the 'testlist01.txt' file, 'trainlist01.txt' can be generated using external preparation scripts. Creating and populating those lists marks the end of the 'TrainTestlist' folder preparation.

Once done with the 'TrainTestlist' folder, one must prepare the 'data_file.csv' file. This CSV file contains, for one given line, the following.

```
<'test' or 'train' status>, <class name>, <file name>, <number of frames>
```

This must be defined for all files and be consistent with information in 'classInd.txt', 'testlist01.txt', 'trainlist01.txt'. Again, this CSV file can be generated using external preparation scripts.

External Preparation Tools

In order to streamline the data preparation process, external preparation scripts can be employed.

Base Training Parameters

The 'train.py' script, located within the root folder, contains key variables that will directly affect training performance.

The 'early_stopper' instantiation of 'EarlyStopping' has a 'patience' field that is intended to stop training when learning stops. This happens after several epochs without validation loss improvement. Setting a high patience will potentially cause overfitting in the long run while setting a low value may prevent the network from learning further and limit model accuracy.

The 'main' function definition holds the main training settings that must be set before running 'train.py'.

- 'model' can be one of 'lstm', 'lrcn', 'mlp', 'conv_3d', 'c3d';
- 'seq_length' should be consistent with actual sequence length;

- ‘batch_size’ should be adequate considering number of training examples;
- ‘nb_epoch’ is a target likely to never be reached considering patience.

Execution

Run ‘train_cnn.py’, then run ‘train.py’. You may use TensorBoard to get live performance results in a convenient web GUI.