Data Formats and Data Loading Options

Training and validation data are specified on the command line with arguments like the following:

```
--train-data "train.pfile,context=5,ignore-label=0:3-9,map-label=1:0/2:1,partition=1000m"
--valid-data "valid.pfile,stream=False,random=True"
```

The part before the first comma (if there is any) specifies the file name. Glob-style wildcards may be used to specify multiple files (currently not supported for Kaldi data files). The data files may be compressed using gzip or bz2, in which case they will have an extension of "`.gz`" or "`.bz2`" in addition to the original extension.

After the file name, you may specify any number of data loading options in the format of "`key=value`". The functions of the options are described in the sections below.

Supported Data Formats

PDNN currently supports three data formats: PFiles, Python pickle files, and Kaldi files.

PFiles

The PFile is the ICSI feature file archive format. PFiles have the extension `.pfile`. A PFile can store multiple *sentences*, each of which is a sequence of *frames*. Each frame is associated with a feature vector and one or more labels. Below is the content of an example PFile.

Sentence ID	Frame ID	Feature Vector	Class Label
0	0	[0.2, 0.3, 0.5, 1.4, 1.8, 2.5]	10
0	1	[1.3, 2.1, 0.3, 0.1, 1.4, 0.9]	179
1	0	[0.3, 0.5, 0.5, 1.4, 0.8, 1.4]	32

For speech processing, sentences and frames correspond to utterances and frames respectively. Frames are indexed within each sentence. For other applications, you can use fake sentence indices and frame indices. For example, with *N* instances, you can set all the sentence indices to 0, and the frame indices to 0, 1, ..., *N*-1.

A standard PFile toolkit is `pfile_utils-v0_51`. This script installs it automatically if you are running Linux. HTK users can convert HTK feature and label files into PFiles using this Python script. Refer to the comments at the top for more information.

Python Pickle Files

Python pickle files may have the extension "`.pickle`" or "`.pkl`". A Python pickle file serializes a tuple of two numpy arrays, `(feature, label)`. There is no notion of "sentences" in pickle files; in other words, a pickle files stores exactly one sentence. `feature` is a 2-D numpy array, where each row is the feature vector of one instance; `label` is a 1-D numpy array, where each element is the class label of one instance.

To read a (gzip-compressed) pickle file in Python:

```
> import cPickle, numpy, gzip
> with gzip.open('filename.pkl.gz', 'rb') as f:
>     feature, label = cPickle.load(f)
```

To create a (gzip-compressed) pickle file in Python:

```
> import cPickle, numpy, gzip
> feature = numpy.array([[0.2, 0.3, 0.5, 1.4], [1.3, 2.1, 0.3, 0.1], [0.3, 0.5, 0.5, 1.4]], dtype = 'float32')
> label = numpy.array([2, 0, 1])
> with gzip.open('filename.pkl.gz', 'wb') as f:
>     cPickle.dump((feature, label), f)
```

Kaldi Files

The Kaldi data files accepted by PDNN are "Kaldi script files" with the extension ".scp". These files contain "pointers" to the actual feature data stored in "Kaldi archive files" ending in ".ark". Each line of a Kaldi script file specifies the name of an *utterance* (equivalent to a *sentence* in pfiles), and its offset in a Kaldi archive file, as follows:

```
utt01 train.ark:15213
```

Labels corresponding to the features are provided by "alignment files" ending in ".ali". To specify an alignment file, use the option "label=filename.ali". Alignment files are plain text files, where each line specifies the name of an utterance, followed by the label of each frame in this utterance. Below is an example:

```
utt01 0 51 51 51 51 51 51 48 48 7 7 7 7 51 51 51 51 48
```

On-The-Fly Context Padding and Label Manipulation

Oftentimes, we want to include the features of neighboring frames into the feature vector of the current frame. Of course this can be done when you prepare the data files, but this will bloat their size. A more clever way is to perform this "context padding" on the fly. PDNN provides the option "context" to do this. Specifying "context=5" will pad each frame with 5 frames on either side, so that the feature vector becomes 11 times the original dimensionality. Specifying "context=5:1" will pad each frame with 5 frames on the left and 1 frame on the right. Alternatively, you can also specify "lcxt=5,rcxt=1". Context padding does not cross sentence boundaries. At the beginning and end of each sentence, the first and last frames are repeated when the context reaches beyond the sentence boundary.

Some frames in the data files may be garbage frames (i.e. they do not belong to any of the classes to be classified), but they are important in making up the context for useful frames. To ignore such frames, you can assign a special class label (say *c*) to these frames, and specify the option "ignore-label=*c*". The garbage frames will be discarded; but the context of neighboring frames will still be correct, as the garbage frames are only discarded **after** context padding happens. Sometimes you may also want to train a classifier for only a subset of the classes in a data file. In such cases, you may specify multiple class labels to be ignored, e.g. "ignore-label=0:2:7-9". Multiple class labels are separated by colons; contiguous class labels may be specified with a dash.

When training a classifier of *N* classes, PDNN requires that their class labels be 0, 1, ..., *N*-1. When you ignore some class labels, the remaining class labels may not form such a sequence. In this situation, you may use the "map-label" option to map the remaining class labels to 0, 1, ..., *N*-1. For example, to map the classes 1, 3, 4, 5, 6 to 0, 1, 2, 3, 4, you can specify "map-label=1:0/3:1/4:2/5:3/6:4". Each pair of labels are separated by a colon; pairs are separated by slashes. The label mapping happens **after** unwanted labels are discarded; all the mappings are applied simultaneously (therefore class 3 is mapped to class 1 and is *not* further mapped to class 0). You may also use this option to merge classes. For example, "map-label=1:0/3:1/4-6:2" will map all the labels 4, 5, 6 to class 2.

Option	Example	Function
context	context=5 context=5:1 lcxt=5,rcxt=1	Pads the feature vector of each frame with specified numbers of frames on either side
ignore-label	ignore-label=0:2:7-9	Discards frames with specified class labels
map-label	map-label=1:0/3:1/4-6:2	Maps class labels so they form a sequence starting from 0; Merges classes

Partitions, Streaming and Shuffling

The training / validation corpus may be too large to fit in the CPU or GPU memory. Therefore they are broken down into several levels of units: files, partitions, and minibatches. Such division happens after context padding and label manipulation, and the concept of "sentences" are no longer relevant. As a result, a sentence may be broken into multiple partitions of minibatches.

Both the training and validation corpora may consist of multiple files that can be matched by a single glob-style pattern. At any point in time, at most one file is held in the CPU memory. This means if you have multiple files, all the files will be reloaded every epoch. This can be very inefficient; you can avoid this inefficiency by lumping all the data into a single file if they can fit in the CPU memory.

A partition is the amount of data that is fed to the GPU at a time. **For pickle files, a partition is always an entire file**; for other files, you may specify the partition size with the option "partition", e.g. "partition=1000m". The partition size is specified in megabytes (2^{20} bytes); the suffix "m" is optional. The default partition size is 600 MB.

Files may be read in either the "stream" or the "non-stream" mode, controlled by the option "`stream=True`" or "`stream=False`". In the non-stream mode, an entire file is kept in the CPU memory. If there is only one file in the training / validation corpus, the file is loaded only once (and this is efficient). In the stream mode, only a partition is kept in the CPU memory. This is useful when the corpus is too large to fit in the CPU memory. **Currently, PFiles can be loaded in either the stream mode or the non-stream mode; pickle files can only be loaded in the non-stream mode; Kaldi files can only be loaded in the stream mode.**

It is usually desirable that instances of different classes be mixed evenly in the training data. To achieve this, you may specify the option "`random=True`". This options shuffles the order of the training instances loaded into the CPU memory at a time: in the stream mode, instances are shuffled partition by partition; in the non-stream mode, instance are shuffled across an entire file. The latter achieves better mixing, so it is again recommended to turn off the stream mode when the files can fit in the CPU memory.

A minibatch is the amount of data consumed by the training procedure between successive updates of the model parameters. The minibatch size is not specified as a data loading option, but as a separate command-line argument to the training scripts. A partition may not consist of a whole number of minibatches; the last instances in each partition that are not enough to make a minibatch are discarded.

Option	Default Value	Function
partition	600m	Specifies the partition size
stream	False	Turns the stream mode on or off
random	False	Shuffles the instances in each file (non-stream mode) or partition (stream mode)