# Edge Machine Learning: Pytorch Library

This package includes PyTorch implementations of following algorithms and training techniques developed as part of EdgeML. The PyTorch graphs for the forward/backward pass of these algorithms are packaged as `edgeml_pytorch.graph` and the trainers for these algorithms are in `edgeml_pytorch.trainer`.

1. Bonsai: `edgeml_pytorch.graph.bonsai` implements the Bonsai prediction graph. The three-phase training routine for Bonsai is decoupled from the forward graph to facilitate a plug and play behaviour wherein Bonsai can be combined with or used as a final layer classifier for other architectures (RNNs, CNNs). See `edgeml_pytorch.trainer.bonsaiTrainer` for 3-phase training.
2. ProtoNN: `edgeml_pytorch.graph.protoNN` implements the ProtoNN prediction functions. The training routine for ProtoNN is decoupled from the forward graph to facilitate a plug and play behaviour wherein ProtoNN can be combined with or used as a final layer classifier for other architectures (RNNs, CNNs). The training routine is implemented in `edgeml_pytorch.trainer.protoNNTrainer`.
3. FastRNN & FastGRNN: `edgeml_pytorch.graph.rnn` provides various RNN cells --- including new cells `FastRNNCell` and `FastGRNNCell` as well as `UGRNNCell`, `GRUCell`, and `LSTMCell` --- with features like low-rank parameterisation of weight matrices and custom non-linearities. Akin to Bonsai and ProtoNN, the three-phase training routine for FastRNN and FastGRNN is decoupled from the custom cells to enable plug and play behaviour of the custom RNN cells in other architectures (NMT, Encoder-Decoder etc.). Additionally, numerically equivalent CUDA-based implementations `FastRNNCUDACell` and `FastGRNNCUDACell` are provided for faster training. `edgeml_pytorch.graph.rnn.Fast(G)RNN(CUDA)` provides unrolled RNNs equivalent to `nn.LSTM` and `nn.GRU`. `edgeml_pytorch.trainer.fastmodel` presents a sample multi-layer RNN + multi-class classifier model.
4. S-RNN: `edgeml_pytorch.graph.rnn.SRNN2` implements a 2 layer SRNN network which can be instantiated with a choice of RNN cell. The training routine for SRNN is in `edgeml_pytorch.trainer.srnnTrainer`.
5. DROCC & DROCC-LF: `edgeml_pytorch.trainer.drocc_trainer` implements a DROCC meta-trainer for training any given model architecture for one-class classification on the supplied dataset. `edgeml_pytorch.trainer.drocclf_trainer` implements the DROCC-LF variant for training models for one-class classification with limited negatives.
6. RNNPool: `edgeml_pytorch.graph.RNNPool` implements the RNNPool pooling layer which can be instantiated with the dimensions of the input patch and the hidden states. Currently only the inference code is implemented, as training routines are written individually for specific use cases. Please checkout the RNNPool examples for reference implementations of the trainer modules.

Usage directions and examples notebooks for this package are provided here.

## Installation

It is highly recommended that EdgeML be installed in a virtual environment. Please create a new virtual environment using your environment manager (virtualenv or Anaconda). Make sure the new environment is active before running the below mentioned commands.

Use pip to install requirements before installing the `edgeml_pytorch` library. Details for cpu based installation and gpu based installation provided below.

## CPU

```
pip install -r requirements-cpu.txt
pip install -e .
```

Tested on Python3.6 with >= PyTorch 1.1.0.

## GPU

Install appropriate CUDA and cuDNN [Tested with >= CUDA 8.1 and cuDNN >= 6.1]

```
pip install -r requirements-gpu.txt
pip install -e .
pip install -e edgeml_pytorch/cuda/
```

**Note**: For using the optimized FastGRNNCUDA implementation, it is recommended to use CUDA v10.1, gcc 7.5 and cuDNN v7.6 and torch==1.4.0. Also, there are some known issues when compiling custom CUDA kernels on Windows pytorch/#11004.