Ensemble-based Low Precision Inference in Software

Description

Implemented based off of Luca Mocerino's Binary-Neural-Networks-PyTorch-1.0, this repository implements the proposed design in my Honours project.

The project is organized as follows:

- **models** folder contains CNN models (simple mlp, Network-in-Network, LeNet5, etc.)
- **classifiers/{type}_classifier.py** contains the test and train procedures; where type = {bnn, xnor, dorefa}
- **models/{type}_layers.py** contains the binarylayers implementation (binary activation, binary conv and fully-connected layers, gradient update); where type = {bnn, xnor, dorefa}
- **yml** folder contains configuration files with hyperparameters
- **main.py** represents the entry file
- **ensemble_algorithms** this folder contains the different ensemble training/testing algorithms that I experimented with, inclusing the main one, dynamic expert voting At the end of the training process, each member's loss and accuracy per epoch graphs are saved at `classifiers/graphs`. This folder is not tracked by git and gets re-written every time the training process is run so ensure that you save them at a permanent location before running any new experiments.

Installation

All packages are in *requirement.txt*
Install the dependencies:

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

Basic usage

```
$ python main.py app:{yml_file}
```

Example

LeNet on CIFAR10 dataset. All hyper parameters are in .yml file.

```
$ python main.py app:yml/lenet_cifar10.yml
```

How to enable/disable perspective transformation?

The code has the functionality to train and test the expert models with perspective transformation enabled to represent the fact that these models will be looking at the object from different angles. Initial results indicate that enabling it should not affect the final accuracy of the ensemble.
To enable perspective transformation, first you need to include the transformation index when loading the test and training datasets for the expert models in `utils.py`, for example:

```
train_loader = dataset.load_train_data
(
  labels=labels,
  cuda=cuda,
  download=download_data,
  batch_size=train_batch_size,
  transformation_type=transformation_type_index,
)
test_loader = dataset.load_test_data
(
  labels=labels,
  cuda=cuda,
  download=download_data,
  batch_size=test_batch_size,
  transformation_type=transformation_type_index,
)
```

Next, you need to include the `transformation_type` when loading the test data for expert models in `dynamic_expert_voting.py`:

```
expert_member_index = 0
  for member in self.ensemble[1:]:
    test_loader = dataset.load_test_data(
      cuda=cuda,
      download=download_data,
      batch_size=test_batch_size,
      transformation_type=expert_member_index,
    )
```

How to switch between running on MNIST and CIFAR10?

1. Ensure your config file has the coorect dataset name, model count, and labels.
2. You need to ensure that the model count matches the number of perspective transformations for the respective dataset.
3. You need to adjust the respective model size and parameter count for the specific dataset. Current settings for LeNet:
   MNIST
   LeNet5

```
self.features = nn.Sequential(
  nn.Conv2d(1, 20, kernel_size=5, stride=1),
  nn.BatchNorm2d(20, eps=1e-4, momentum=0.1, affine=False),
  nn.ReLU(inplace=True),
  nn.MaxPool2d(kernel_size=2, stride=2),
  XNORConv2d(20, 50, kernel_size=5, stride=1, padding=0),
  nn.MaxPool2d(kernel_size=2, stride=2),
  nn.Flatten(),
)
self.classifier = nn.Sequential(
  BNLinearReLU(800, 500),
  nn.BatchNorm1d(500, eps=1e-4, momentum=0.1, affine=False),
  nn.Linear(500, out_classes),
)
```

LeNet Expert

```
self.features = nn.Sequential(
  nn.Conv2d(1, 10, kernel_size=5, stride=1),
  nn.BatchNorm2d(10, eps=1e-4, momentum=0.1, affine=False),
  nn.ReLU(inplace=True),
  nn.MaxPool2d(kernel_size=2, stride=2),
  XNORConv2d(10, 25, kernel_size=5, stride=1, padding=0),
  nn.MaxPool2d(kernel_size=2, stride=2),
  nn.Flatten(),
)
self.classifier = nn.Sequential(
  BNLinearReLU(400, 250),
  nn.BatchNorm1d(250, eps=1e-4, momentum=0.1, affine=False),
  nn.Linear(250, out_classes),
)
```

CIFAR10

LeNet5

```
self.features = nn.Sequential(
  nn.Conv2d(3, 32, kernel_size=5, stride=1, padding=2),
  nn.BatchNorm2d(32, eps=1e-4, momentum=0.1, affine=False),
  nn.ReLU(inplace=True),
  nn.MaxPool2d(kernel_size=2, stride=2),
  XNORConv2d(32, 64, kernel_size=5, stride=1, padding=2),
  nn.MaxPool2d(kernel_size=2, stride=2),
  nn.Flatten(),
)
self.classifier = nn.Sequential(
  BNLinearReLU(4096, 500),
  nn.BatchNorm1d(500, eps=1e-4, momentum=0.1, affine=False),
  nn.Linear(500, out_classes),
)
```

LeNet Expert

```
self.features = nn.Sequential(
  nn.Conv2d(3, 10, kernel_size=5, stride=1),
  nn.BatchNorm2d(10, eps=1e-4, momentum=0.1, affine=False),
  nn.ReLU(inplace=True),
  nn.MaxPool2d(kernel_size=2, stride=2),
  XNORConv2d(10, 25, kernel_size=5, stride=1, padding=0),
  nn.MaxPool2d(kernel_size=2, stride=2),
  nn.Flatten(),
)
self.classifier = nn.Sequential(
  BNLinearReLU(400, 250),
  nn.BatchNorm1d(250, eps=1e-4, momentum=0.1, affine=False),
  nn.Linear(250, out_classes),
)
```