
MNIST Digit Classification with Lava

***Motivation**: In this tutorial, we will build a Lava Process for an MNIST classifier, using the Lava Processes for LIF neurons and Dense connectivity. The tutorial is useful to get started with Lava in a few minutes.*

This tutorial assumes that you:

- have the Lava framework installed
- are familiar with the Process concept in Lava

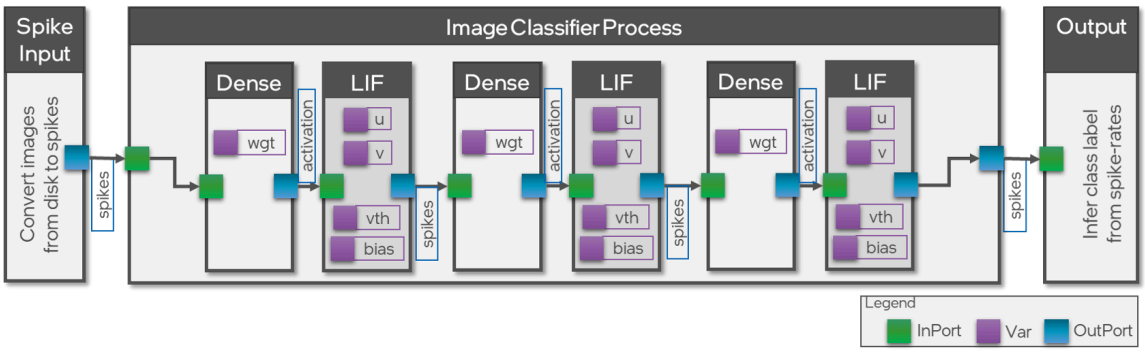
This tutorial gives a bird's-eye view of

- how Lava Process(es) can perform the MNIST digit classification task using Leaky Integrate-and-Fire (LIF) neurons and Dense (fully connected) connectivity.
- how to create a Process
- how to create Python ProcessModels
- how to connect Processes
- how to execute them

Our MNIST Classifier

In this tutorial, we will build a multi-layer feed-forward classifier without any convolutional layers. The architecture is shown below.

> **Important Note**:
>
> The classifier is a simple feed-forward model using pre-trained network parameters (weights and biases). It illustrates how to build, compile and run a functional model in Lava. Please refer to Lava-DL to understand how to train deep networks with Lava.



The 3 Processes shown above are:

- *SpikeInput* Process - generates spikes via integrate and fire dynamics, using the image input
- *ImageClassifier* Process - encapsulates feed-forward architecture of Dense connectivity and LIF neurons
- *Output* Process - accumulates output spikes from the feed-forward process and infers the class label

General Imports

In [1]:
```
import os
import numpy as np
import typing as ty
```

Lava Processes

Below we create the Lava Process classes. We need to define only the structure of the process here. The details about how the Process will be executed are specified in the ProcessModels below.

As mentioned above, we define Processes for

- converting input images to binary spikes from those biases (*SpikeInput*),
- the 3-layer fully connected feed-forward network (*MnistClassifier*)
- accumulating the output spikes and inferring the class for an input image (*OutputProcess*)

In [2]:
```
# Import Process level primitives
from lava.magma.core.process.process import AbstractProcess
from lava.magma.core.process.variable import Var
from lava.magma.core.process.ports.ports import InPort, OutPort
```

In [3]:
```
class SpikeInput(AbstractProcess):
    """Reads image data from the MNIST dataset and converts it to spikes.
    The resulting spike rate is proportional to the pixel value."""

    def __init__(self,
                 vth: int,
                 num_images: ty.Optional[int] = 50, #change this to be 25,50,100
```

```
                 num_steps_per_image: ty.Optional[int] = 128):
        super().__init__()
        shape = (784,)
        self.spikes_out = OutPort(shape=shape)  # Input spikes to the classifier
        self.label_out = OutPort(shape=(1,))  # Ground truth labels to OutputProc
        self.num_images = Var(shape=(1,), init=num_images)
        self.num_steps_per_image = Var(shape=(1,), init=num_steps_per_image)
        self.input_img = Var(shape=shape)
        self.ground_truth_label = Var(shape=(1,))
        self.v = Var(shape=shape, init=0)
        self.vth = Var(shape=(1,), init=vth)


class ImageClassifier(AbstractProcess):
    """A 3 layer feed-forward network with LIF and Dense Processes."""

    def __init__(self, trained_weights_path: str):
        super().__init__()

        # Using pre-trained weights and biases
        real_path_trained_wgts = os.path.realpath(trained_weights_path)

        wb_list = np.load(real_path_trained_wgts, encoding='latin1', allow_pickle=T
        w0 = wb_list[0].transpose().astype(np.int32)
        w1 = wb_list[2].transpose().astype(np.int32)
        w2 = wb_list[4].transpose().astype(np.int32)
        b1 = wb_list[1].astype(np.int32)
        b2 = wb_list[3].astype(np.int32)
        b3 = wb_list[5].astype(np.int32)

        self.spikes_in = InPort(shape=(w0.shape[1],))
        self.spikes_out = OutPort(shape=(w2.shape[0],))
        self.w_dense0 = Var(shape=w0.shape, init=w0)
        self.b_lif1 = Var(shape=(w0.shape[0],), init=b1)
        self.w_dense1 = Var(shape=w1.shape, init=w1)
        self.b_lif2 = Var(shape=(w1.shape[0],), init=b2)
        self.w_dense2 = Var(shape=w2.shape, init=w2)
        self.b_output_lif = Var(shape=(w2.shape[0],), init=b3)

        # Up-level currents and voltages of LIF Processes
        # for resetting (see at the end of the tutorial)
        self.lif1_u = Var(shape=(w0.shape[0],), init=0)
        self.lif1_v = Var(shape=(w0.shape[0],), init=0)
        self.lif2_u = Var(shape=(w1.shape[0],), init=0)
        self.lif2_v = Var(shape=(w1.shape[0],), init=0)
        self.oplif_u = Var(shape=(w2.shape[0],), init=0)
        self.oplif_v = Var(shape=(w2.shape[0],), init=0)


class OutputProcess(AbstractProcess):
    """Process to gather spikes from 10 output LIF neurons and interpret the
    highest spiking rate as the classifier output"""

    def __init__(self, **kwargs):
        super().__init__()
        shape = (10,)
```

```
        n_img = kwargs.pop('num_images', 50)
        self.num_images = Var(shape=(1,), init=n_img)
        self.spikes_in = InPort(shape=shape)
        self.label_in = InPort(shape=(1,))
        self.spikes_accum = Var(shape=shape)  # Accumulated spikes for classificati
        self.num_steps_per_image = Var(shape=(1,), init=128)
        self.pred_labels = Var(shape=(n_img,))
        self.gt_labels = Var(shape=(n_img,))
```

ProcessModels for Python execution

In [4]:
```
# Import parent classes for ProcessModels
from lava.magma.core.model.sub.model import AbstractSubProcessModel
from lava.magma.core.model.py.model import PyLoihiProcessModel

# Import ProcessModel ports, data-types
from lava.magma.core.model.py.ports import PyInPort, PyOutPort
from lava.magma.core.model.py.type import LavaPyType

# Import execution protocol and hardware resources
from lava.magma.core.sync.protocols.loihi_protocol import LoihiProtocol
from lava.magma.core.resources import CPU

# Import decorators
from lava.magma.core.decorator import implements, requires

# Import MNIST dataset
from lava.utils.dataloader.mnist import MnistDataset
np.set_printoptions(linewidth=np.inf)
```

Decorators for ProcessModels: the argument `proc`. Using `protocol` argument, we will specify the

- `@implements`: associates a ProcessModel with a Process through

synchronization protocol used by the ProcessModel. In this tutorial, all ProcessModels execute according to the `LoihiProtocol`. Which means, similar to the Loihi chip, each time-step is divided into *spiking*, *pre-management*, *post-management*, and *learning* phases. It is necessary to specify behaviour of a ProcessModel during the spiking phase using `run_spk` function. Other phases are optional.

- `@requires`: specifies the hardware resource on which a ProcessModel will be executed. In this tutorial, we will execute all ProcessModels on a CPU.

SpikingInput ProcessModel

In [5]:
```
@implements(proc=SpikeInput, protocol=LoihiProtocol)
@requires(CPU)
class PySpikeInputModel(PyLoihiProcessModel):
    num_images: int = LavaPyType(int, int, precision=32)
    spikes_out: PyOutPort = LavaPyType(PyOutPort.VEC_DENSE, bool, precision=1)
    label_out: PyOutPort = LavaPyType(PyOutPort.VEC_DENSE, np.int32,
```

```
                                        precision=32)
    num_steps_per_image: int = LavaPyType(int, int, precision=32)
    input_img: np.ndarray = LavaPyType(np.ndarray, int, precision=32)
    ground_truth_label: int = LavaPyType(int, int, precision=32)
    v: np.ndarray = LavaPyType(np.ndarray, int, precision=32)
    vth: int = LavaPyType(int, int, precision=32)

    def __init__(self, proc_params):
        super().__init__(proc_params=proc_params)
        self.mnist_dataset = MnistDataset()
        self.curr_img_id = 0

    def post_guard(self):
        """Guard function for PostManagement phase.
        """
        if self.time_step % self.num_steps_per_image == 1:
            return True
        return False

    def run_post_mgmt(self):
        """Post-Management phase: executed only when guard function above
        returns True.
        """
        img = self.mnist_dataset.images[self.curr_img_id]
        self.ground_truth_label = self.mnist_dataset.labels[self.curr_img_id]
        self.input_img = img.astype(np.int32) - 127
        self.v = np.zeros(self.v.shape)
        self.label_out.send(np.array([self.ground_truth_label]))
        self.curr_img_id += 1

    def run_spk(self):
        """Spiking phase: executed unconditionally at every time-step
        """
        self.v[:] = self.v + self.input_img
        s_out = self.v > self.vth
        self.v[s_out] = 0  # reset voltage to 0 after a spike
        self.spikes_out.send(s_out)
```

ImageClassifier ProcessModel

Notice that the following process model is further decomposed into sub-Processes, which implement LIF neural dynamics and Dense connectivity. We will not go into the details of how these are implemented in this tutorial.

Also notice that a *SubProcessModel* does not actually contain any concrete execution. This is handled by the ProcessModels of the constituent Processes.

In [6]:
```
from lava.proc.lif.process import LIF
from lava.proc.dense.process import Dense

@implements(ImageClassifier)
@requires(CPU)
class PyImageClassifierModel(AbstractSubProcessModel):
    def __init__(self, proc):
```

```
        self.dense0 = Dense(weights=proc.w_dense0.init)
        self.lif1 = LIF(shape=(64,), bias_mant=proc.b_lif1.init, vth=400,
                        dv=0, du=4095)
        self.dense1 = Dense(weights=proc.w_dense1.init)
        self.lif2 = LIF(shape=(64,), bias_mant=proc.b_lif2.init, vth=350,
                        dv=0, du=4095)
        self.dense2 = Dense(weights=proc.w_dense2.init)
        self.output_lif = LIF(shape=(10,), bias_mant=proc.b_output_lif.init,
                              vth=1, dv=0, du=4095)

        proc.spikes_in.connect(self.dense0.s_in)
        self.dense0.a_out.connect(self.lif1.a_in)
        self.lif1.s_out.connect(self.dense1.s_in)
        self.dense1.a_out.connect(self.lif2.a_in)
        self.lif2.s_out.connect(self.dense2.s_in)
        self.dense2.a_out.connect(self.output_lif.a_in)
        self.output_lif.s_out.connect(proc.spikes_out)

        # Create aliases of SubProcess variables
        proc.lif1_u.alias(self.lif1.u)
        proc.lif1_v.alias(self.lif1.v)
        proc.lif2_u.alias(self.lif2.u)
        proc.lif2_v.alias(self.lif2.v)
        proc.oplif_u.alias(self.output_lif.u)
        proc.oplif_v.alias(self.output_lif.v)
```

OutputProcess ProcessModel

In [7]:

```
@implements(proc=OutputProcess, protocol=LoihiProtocol)
@requires(CPU)
class PyOutputProcessModel(PyLoihiProcessModel):
    label_in: PyInPort = LavaPyType(PyInPort.VEC_DENSE, int, precision=32)
    spikes_in: PyInPort = LavaPyType(PyInPort.VEC_DENSE, bool, precision=1)
    num_images: int = LavaPyType(int, int, precision=32)
    spikes_accum: np.ndarray = LavaPyType(np.ndarray, np.int32, precision=32)
    num_steps_per_image: int = LavaPyType(int, int, precision=32)
    pred_labels: np.ndarray = LavaPyType(np.ndarray, int, precision=32)
    gt_labels: np.ndarray = LavaPyType(np.ndarray, int, precision=32)

    def __init__(self, proc_params):
        super().__init__(proc_params=proc_params)
        self.current_img_id = 0

    def post_guard(self):
        """Guard function for PostManagement phase.
        """
        if self.time_step % self.num_steps_per_image == 0 and \
                self.time_step > 1:
            return True
        return False

    def run_post_mgmt(self):
        """Post-Management phase: executed only when guard function above
        returns True.
        """
```

```
        gt_label = self.label_in.recv()
        pred_label = np.argmax(self.spikes_accum)
        self.gt_labels[self.current_img_id] = gt_label
        self.pred_labels[self.current_img_id] = pred_label
        self.current_img_id += 1
        self.spikes_accum = np.zeros_like(self.spikes_accum)

    def run_spk(self):
        """Spiking phase: executed unconditionally at every time-step
        """
        spk_in = self.spikes_in.recv()
        self.spikes_accum = self.spikes_accum + spk_in
```

Connecting Processes

In [8]:
```
num_images = 50 #change this to be 25,50,100
num_steps_per_image = 128

# Create Process instances
spike_input = SpikeInput(vth=1,
                         num_images=num_images,
                         num_steps_per_image=num_steps_per_image)
mnist_clf = ImageClassifier(
    trained_weights_path=os.path.join('.', 'mnist_pretrained.npy'))
output_proc = OutputProcess(num_images=num_images)

# Connect Processes
spike_input.spikes_out.connect(mnist_clf.spikes_in)
mnist_clf.spikes_out.connect(output_proc.spikes_in)
# Connect Input directly to Output for ground truth labels
spike_input.label_out.connect(output_proc.label_in)
```

If you receive an `UnpicklingError` when instantiating the `ImageClassifier`, make sure to download the pretrained weights from GitHub LFS in the current directory using:

```
git lfs fetch
   git lfs pull
```

Execution and results

Below, we will run the classifier process in a loop of `num_images` number of iterations. Each iteration will run the Process for `num_steps_per_image` number of time-steps.

We take this approach to clear the neural states of all three LIF layers inside the classifier after every image. We need to clear the neural states, because the network parameters were trained assuming clear neural states for each inference.

> Note: Below we have used `Var.set()` function to set the values of internal state variables. The same behaviour can be achieved by using `RefPorts`. See the RefPorts tutorial to learn more about how to use `RefPorts` to access internal state variables of Lava Processes.

In [ ]:
```
from lava.magma.core.run_conditions import RunSteps
from lava.magma.core.run_configs import Loihi1SimCfg

# Loop over all images
for img_id in range(num_images):
    print(f"\rCurrent image: {img_id+1}", end="")

    # Run each image-inference for fixed number of steps
    mnist_clf.run(
        condition=RunSteps(num_steps=num_steps_per_image),
        run_cfg=Loihi1SimCfg(select_sub_proc_model=True,
                             select_tag='fixed_pt'))

    # Reset internal neural state of LIF neurons
    mnist_clf.lif1_u.set(np.zeros((64,), dtype=np.int32))
    mnist_clf.lif1_v.set(np.zeros((64,), dtype=np.int32))
    mnist_clf.lif2_u.set(np.zeros((64,), dtype=np.int32))
    mnist_clf.lif2_v.set(np.zeros((64,), dtype=np.int32))
    mnist_clf.oplif_u.set(np.zeros((10,), dtype=np.int32))
    mnist_clf.oplif_v.set(np.zeros((10,), dtype=np.int32))

# Gather ground truth and predictions before stopping exec
ground_truth = output_proc.gt_labels.get().astype(np.int32)
predictions = output_proc.pred_labels.get().astype(np.int32)

# Stop the execution
mnist_clf.stop()

accuracy = np.sum(ground_truth==predictions)/ground_truth.size * 100

print(f"\nGround truth: {ground_truth}\n"
      f"Predictions : {predictions}\n"
      f"Accuracy    : {accuracy}")
```

```
Current image: 1
/tmp/ipykernel_19906/4202512221.py:30: DeprecationWarning: Conversion of an array wi
th ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract
a single element from your array before performing this operation. (Deprecated NumPy
1.25.)
  self.gt_labels[self.current_img_id] = gt_label
Current image: 17
```

> **Important Note**:
>
> The classifier is a simple feed-forward model using pre-trained network parameters (weights and biases). It illustrates how to build, compile and run a functional model in Lava. Please refer to Lava-DL to understand how to train deep networks with Lava.

How to learn more?

Follow the links below for deep-dive tutorials on the concepts in this tutorial:

- Processes
- ProcessModel
- Execution
- SubProcessModels or Hierarchical Processes
- RefPorts

If you want to find out more about Lava, have a look at the Lava documentation or dive into the source code.

To receive regular updates on the latest developments and releases of the Lava Software Framework please subscribe to the INRC newsletter.

In [ ]:
```
import os
import numpy as np
import typing as ty

# Import Process level primitives
from lava.magma.core.process.process import AbstractProcess
from lava.magma.core.process.variable import Var
from lava.magma.core.process.ports.ports import InPort, OutPort

class SpikeInput(AbstractProcess):
    """Reads image data from the MNIST dataset and converts it to spikes.
    The resulting spike rate is proportional to the pixel value."""

    def __init__(self,
                 vth: int,
                 num_images: ty.Optional[int] = 25, #change this to be 25,50,100
                 num_steps_per_image: ty.Optional[int] = 128):
        super().__init__()
        shape = (784,)
        self.spikes_out = OutPort(shape=shape)  # Input spikes to the classifier
        self.label_out = OutPort(shape=(1,))  # Ground truth labels to OutputProc
        self.num_images = Var(shape=(1,), init=num_images)
        self.num_steps_per_image = Var(shape=(1,), init=num_steps_per_image)
        self.input_img = Var(shape=shape)
        self.ground_truth_label = Var(shape=(1,))
        self.v = Var(shape=shape, init=0)
        self.vth = Var(shape=(1,), init=vth)


class ImageClassifier(AbstractProcess):
    """A 3 layer feed-forward network with LIF and Dense Processes."""

    def __init__(self, trained_weights_path: str):
        super().__init__()

        # Using pre-trained weights and biases
        real_path_trained_wgts = os.path.realpath(trained_weights_path)
```

```
        wb_list = np.load(real_path_trained_wgts, encoding='latin1', allow_pickle=T
        w0 = wb_list[0].transpose().astype(np.int32)
        w1 = wb_list[2].transpose().astype(np.int32)
        w2 = wb_list[4].transpose().astype(np.int32)
        b1 = wb_list[1].astype(np.int32)
        b2 = wb_list[3].astype(np.int32)
        b3 = wb_list[5].astype(np.int32)

        self.spikes_in = InPort(shape=(w0.shape[1],))
        self.spikes_out = OutPort(shape=(w2.shape[0],))
        self.w_dense0 = Var(shape=w0.shape, init=w0)
        self.b_lif1 = Var(shape=(w0.shape[0],), init=b1)
        self.w_dense1 = Var(shape=w1.shape, init=w1)
        self.b_lif2 = Var(shape=(w1.shape[0],), init=b2)
        self.w_dense2 = Var(shape=w2.shape, init=w2)
        self.b_output_lif = Var(shape=(w2.shape[0],), init=b3)

        # Up-level currents and voltages of LIF Processes
        # for resetting (see at the end of the tutorial)
        self.lif1_u = Var(shape=(w0.shape[0],), init=0)
        self.lif1_v = Var(shape=(w0.shape[0],), init=0)
        self.lif2_u = Var(shape=(w1.shape[0],), init=0)
        self.lif2_v = Var(shape=(w1.shape[0],), init=0)
        self.oplif_u = Var(shape=(w2.shape[0],), init=0)
        self.oplif_v = Var(shape=(w2.shape[0],), init=0)


class OutputProcess(AbstractProcess):
    """Process to gather spikes from 10 output LIF neurons and interpret the
    highest spiking rate as the classifier output"""

    def __init__(self, **kwargs):
        super().__init__()
        shape = (10,)
        n_img = kwargs.pop('num_images', 25)
        self.num_images = Var(shape=(1,), init=n_img)
        self.spikes_in = InPort(shape=shape)
        self.label_in = InPort(shape=(1,))
        self.spikes_accum = Var(shape=shape)  # Accumulated spikes for classificati
        self.num_steps_per_image = Var(shape=(1,), init=128)
        self.pred_labels = Var(shape=(n_img,))
        self.gt_labels = Var(shape=(n_img,))

# Import parent classes for ProcessModels
from lava.magma.core.model.sub.model import AbstractSubProcessModel
from lava.magma.core.model.py.model import PyLoihiProcessModel

# Import ProcessModel ports, data-types
from lava.magma.core.model.py.ports import PyInPort, PyOutPort
from lava.magma.core.model.py.type import LavaPyType

# Import execution protocol and hardware resources
from lava.magma.core.sync.protocols.loihi_protocol import LoihiProtocol
from lava.magma.core.resources import CPU

# Import decorators
```

```
from lava.magma.core.decorator import implements, requires

# Import MNIST dataset
from lava.utils.dataloader.mnist import MnistDataset
np.set_printoptions(linewidth=np.inf)

@implements(proc=SpikeInput, protocol=LoihiProtocol)
@requires(CPU)
class PySpikeInputModel(PyLoihiProcessModel):
    num_images: int = LavaPyType(int, int, precision=32)
    spikes_out: PyOutPort = LavaPyType(PyOutPort.VEC_DENSE, bool, precision=1)
    label_out: PyOutPort = LavaPyType(PyOutPort.VEC_DENSE, np.int32,
                                      precision=32)
    num_steps_per_image: int = LavaPyType(int, int, precision=32)
    input_img: np.ndarray = LavaPyType(np.ndarray, int, precision=32)
    ground_truth_label: int = LavaPyType(int, int, precision=32)
    v: np.ndarray = LavaPyType(np.ndarray, int, precision=32)
    vth: int = LavaPyType(int, int, precision=32)

    def __init__(self, proc_params):
        super().__init__(proc_params=proc_params)
        self.mnist_dataset = MnistDataset()
        self.curr_img_id = 0

    def post_guard(self):
        """Guard function for PostManagement phase.
        """
        if self.time_step % self.num_steps_per_image == 1:
            return True
        return False

    def run_post_mgmt(self):
        """Post-Management phase: executed only when guard function above
        returns True.
        """
        img = self.mnist_dataset.images[self.curr_img_id]
        self.ground_truth_label = self.mnist_dataset.labels[self.curr_img_id]
        self.input_img = img.astype(np.int32) - 127
        self.v = np.zeros(self.v.shape)
        self.label_out.send(np.array([self.ground_truth_label]))
        self.curr_img_id += 1

    def run_spk(self):
        """Spiking phase: executed unconditionally at every time-step
        """
        self.v[:] = self.v + self.input_img
        s_out = self.v > self.vth
        self.v[s_out] = 0  # reset voltage to 0 after a spike
        self.spikes_out.send(s_out)

from lava.proc.lif.process import LIF
from lava.proc.dense.process import Dense

@implements(ImageClassifier)
@requires(CPU)
class PyImageClassifierModel(AbstractSubProcessModel):
```

```
    def __init__(self, proc):
        self.dense0 = Dense(weights=proc.w_dense0.init)
        self.lif1 = LIF(shape=(64,), bias_mant=proc.b_lif1.init, vth=400,
                        dv=0, du=4095)
        self.dense1 = Dense(weights=proc.w_dense1.init)
        self.lif2 = LIF(shape=(64,), bias_mant=proc.b_lif2.init, vth=350,
                        dv=0, du=4095)
        self.dense2 = Dense(weights=proc.w_dense2.init)
        self.output_lif = LIF(shape=(10,), bias_mant=proc.b_output_lif.init,
                              vth=1, dv=0, du=4095)

        proc.spikes_in.connect(self.dense0.s_in)
        self.dense0.a_out.connect(self.lif1.a_in)
        self.lif1.s_out.connect(self.dense1.s_in)
        self.dense1.a_out.connect(self.lif2.a_in)
        self.lif2.s_out.connect(self.dense2.s_in)
        self.dense2.a_out.connect(self.output_lif.a_in)
        self.output_lif.s_out.connect(proc.spikes_out)

        # Create aliases of SubProcess variables
        proc.lif1_u.alias(self.lif1.u)
        proc.lif1_v.alias(self.lif1.v)
        proc.lif2_u.alias(self.lif2.u)
        proc.lif2_v.alias(self.lif2.v)
        proc.oplif_u.alias(self.output_lif.u)
        proc.oplif_v.alias(self.output_lif.v)

num_images = 50 #change this to be 25,50,100
num_steps_per_image = 128

# Create Process instances
spike_input = SpikeInput(vth=1,
                         num_images=num_images,
                         num_steps_per_image=num_steps_per_image)
mnist_clf = ImageClassifier(
    trained_weights_path=os.path.join('.', 'mnist_pretrained.npy'))
output_proc = OutputProcess(num_images=num_images)

# Connect Processes
spike_input.spikes_out.connect(mnist_clf.spikes_in)
mnist_clf.spikes_out.connect(output_proc.spikes_in)
# Connect Input directly to Output for ground truth labels
spike_input.label_out.connect(output_proc.label_in)
```

In [ ]:

```
from lava.magma.core.run_conditions import RunSteps
from lava.magma.core.run_configs import Loihi1SimCfg

# Loop over all images
for img_id in range(num_images):
    print(f"\rCurrent image: {img_id+1}", end="")

    # Run each image-inference for fixed number of steps
    mnist_clf.run(
        condition=RunSteps(num_steps=num_steps_per_image),
        run_cfg=Loihi1SimCfg(select_sub_proc_model=True,
                             select_tag='fixed_pt'))
```

```
    # Reset internal neural state of LIF neurons
    mnist_clf.lif1_u.set(np.zeros((64,), dtype=np.int32))
    mnist_clf.lif1_v.set(np.zeros((64,), dtype=np.int32))
    mnist_clf.lif2_u.set(np.zeros((64,), dtype=np.int32))
    mnist_clf.lif2_v.set(np.zeros((64,), dtype=np.int32))
    mnist_clf.oplif_u.set(np.zeros((10,), dtype=np.int32))
    mnist_clf.oplif_v.set(np.zeros((10,), dtype=np.int32))

# Gather ground truth and predictions before stopping exec
ground_truth = output_proc.gt_labels.get().astype(np.int32)
predictions = output_proc.pred_labels.get().astype(np.int32)

# Stop the execution
mnist_clf.stop()

accuracy = np.sum(ground_truth==predictions)/ground_truth.size * 100

print(f"\nGround truth: {ground_truth}\n"
      f"Predictions : {predictions}\n"
      f"Accuracy    : {accuracy}")
```

In [ ]:

In [ ]:
```
import os
import numpy as np
import typing as ty

# Import Process level primitives
from lava.magma.core.process.process import AbstractProcess
from lava.magma.core.process.variable import Var
from lava.magma.core.process.ports.ports import InPort, OutPort

class SpikeInput(AbstractProcess):
    """Reads image data from the MNIST dataset and converts it to spikes.
    The resulting spike rate is proportional to the pixel value."""

    def __init__(self,
                 vth: int,
                 num_images: ty.Optional[int] = 25, #change this to be 25,50,100
                 num_steps_per_image: ty.Optional[int] = 128):
        super().__init__()
        shape = (784,)
        self.spikes_out = OutPort(shape=shape)  # Input spikes to the classifier
        self.label_out = OutPort(shape=(1,))  # Ground truth labels to OutputProc
        self.num_images = Var(shape=(1,), init=num_images)
        self.num_steps_per_image = Var(shape=(1,), init=num_steps_per_image)
        self.input_img = Var(shape=shape)
        self.ground_truth_label = Var(shape=(1,))
        self.v = Var(shape=shape, init=0)
        self.vth = Var(shape=(1,), init=vth)


class ImageClassifier(AbstractProcess):
    """A 3 layer feed-forward network with LIF and Dense Processes."""
```

```
    def __init__(self, trained_weights_path: str):
        super().__init__()

        # Using pre-trained weights and biases
        real_path_trained_wgts = os.path.realpath(trained_weights_path)

        wb_list = np.load(real_path_trained_wgts, encoding='latin1', allow_pickle=T
        w0 = wb_list[0].transpose().astype(np.int32)
        w1 = wb_list[2].transpose().astype(np.int32)
        w2 = wb_list[4].transpose().astype(np.int32)
        b1 = wb_list[1].astype(np.int32)
        b2 = wb_list[3].astype(np.int32)
        b3 = wb_list[5].astype(np.int32)

        self.spikes_in = InPort(shape=(w0.shape[1],))
        self.spikes_out = OutPort(shape=(w2.shape[0],))
        self.w_dense0 = Var(shape=w0.shape, init=w0)
        self.b_lif1 = Var(shape=(w0.shape[0],), init=b1)
        self.w_dense1 = Var(shape=w1.shape, init=w1)
        self.b_lif2 = Var(shape=(w1.shape[0],), init=b2)
        self.w_dense2 = Var(shape=w2.shape, init=w2)
        self.b_output_lif = Var(shape=(w2.shape[0],), init=b3)

        # Up-level currents and voltages of LIF Processes
        # for resetting (see at the end of the tutorial)
        self.lif1_u = Var(shape=(w0.shape[0],), init=0)
        self.lif1_v = Var(shape=(w0.shape[0],), init=0)
        self.lif2_u = Var(shape=(w1.shape[0],), init=0)
        self.lif2_v = Var(shape=(w1.shape[0],), init=0)
        self.oplif_u = Var(shape=(w2.shape[0],), init=0)
        self.oplif_v = Var(shape=(w2.shape[0],), init=0)


class OutputProcess(AbstractProcess):
    """Process to gather spikes from 10 output LIF neurons and interpret the
    highest spiking rate as the classifier output"""

    def __init__(self, **kwargs):
        super().__init__()
        shape = (10,)
        n_img = kwargs.pop('num_images', 25)
        self.num_images = Var(shape=(1,), init=n_img)
        self.spikes_in = InPort(shape=shape)
        self.label_in = InPort(shape=(1,))
        self.spikes_accum = Var(shape=shape)  # Accumulated spikes for classificati
        self.num_steps_per_image = Var(shape=(1,), init=128)
        self.pred_labels = Var(shape=(n_img,))
        self.gt_labels = Var(shape=(n_img,))

# Import parent classes for ProcessModels
from lava.magma.core.model.sub.model import AbstractSubProcessModel
from lava.magma.core.model.py.model import PyLoihiProcessModel

# Import ProcessModel ports, data-types
from lava.magma.core.model.py.ports import PyInPort, PyOutPort
```

```
from lava.magma.core.model.py.type import LavaPyType

# Import execution protocol and hardware resources
from lava.magma.core.sync.protocols.loihi_protocol import LoihiProtocol
from lava.magma.core.resources import CPU

# Import decorators
from lava.magma.core.decorator import implements, requires

# Import MNIST dataset
from lava.utils.dataloader.mnist import MnistDataset
np.set_printoptions(linewidth=np.inf)

@implements(proc=SpikeInput, protocol=LoihiProtocol)
@requires(CPU)
class PySpikeInputModel(PyLoihiProcessModel):
    num_images: int = LavaPyType(int, int, precision=32)
    spikes_out: PyOutPort = LavaPyType(PyOutPort.VEC_DENSE, bool, precision=1)
    label_out: PyOutPort = LavaPyType(PyOutPort.VEC_DENSE, np.int32,
                                      precision=32)
    num_steps_per_image: int = LavaPyType(int, int, precision=32)
    input_img: np.ndarray = LavaPyType(np.ndarray, int, precision=32)
    ground_truth_label: int = LavaPyType(int, int, precision=32)
    v: np.ndarray = LavaPyType(np.ndarray, int, precision=32)
    vth: int = LavaPyType(int, int, precision=32)

    def __init__(self, proc_params):
        super().__init__(proc_params=proc_params)
        self.mnist_dataset = MnistDataset()
        self.curr_img_id = 0

    def post_guard(self):
        """Guard function for PostManagement phase.
        """
        if self.time_step % self.num_steps_per_image == 1:
            return True
        return False

    def run_post_mgmt(self):
        """Post-Management phase: executed only when guard function above
        returns True.
        """
        img = self.mnist_dataset.images[self.curr_img_id]
        self.ground_truth_label = self.mnist_dataset.labels[self.curr_img_id]
        self.input_img = img.astype(np.int32) - 127
        self.v = np.zeros(self.v.shape)
        self.label_out.send(np.array([self.ground_truth_label]))
        self.curr_img_id += 1

    def run_spk(self):
        """Spiking phase: executed unconditionally at every time-step
        """
        self.v[:] = self.v + self.input_img
        s_out = self.v > self.vth
        self.v[s_out] = 0  # reset voltage to 0 after a spike
        self.spikes_out.send(s_out)
```

```
from lava.proc.lif.process import LIF
from lava.proc.dense.process import Dense

@implements(ImageClassifier)
@requires(CPU)
class PyImageClassifierModel(AbstractSubProcessModel):
    def __init__(self, proc):
        self.dense0 = Dense(weights=proc.w_dense0.init)
        self.lif1 = LIF(shape=(64,), bias_mant=proc.b_lif1.init, vth=400,
                        dv=0, du=4095)
        self.dense1 = Dense(weights=proc.w_dense1.init)
        self.lif2 = LIF(shape=(64,), bias_mant=proc.b_lif2.init, vth=350,
                        dv=0, du=4095)
        self.dense2 = Dense(weights=proc.w_dense2.init)
        self.output_lif = LIF(shape=(10,), bias_mant=proc.b_output_lif.init,
                              vth=1, dv=0, du=4095)

        proc.spikes_in.connect(self.dense0.s_in)
        self.dense0.a_out.connect(self.lif1.a_in)
        self.lif1.s_out.connect(self.dense1.s_in)
        self.dense1.a_out.connect(self.lif2.a_in)
        self.lif2.s_out.connect(self.dense2.s_in)
        self.dense2.a_out.connect(self.output_lif.a_in)
        self.output_lif.s_out.connect(proc.spikes_out)

        # Create aliases of SubProcess variables
        proc.lif1_u.alias(self.lif1.u)
        proc.lif1_v.alias(self.lif1.v)
        proc.lif2_u.alias(self.lif2.u)
        proc.lif2_v.alias(self.lif2.v)
        proc.oplif_u.alias(self.output_lif.u)
        proc.oplif_v.alias(self.output_lif.v)

num_images = 25 #change this to be 25,50,100
num_steps_per_image = 128

# Create Process instances
spike_input = SpikeInput(vth=1,
                         num_images=num_images,
                         num_steps_per_image=num_steps_per_image)
mnist_clf = ImageClassifier(
    trained_weights_path=os.path.join('.', 'mnist_pretrained.npy'))
output_proc = OutputProcess(num_images=num_images)

# Connect Processes
spike_input.spikes_out.connect(mnist_clf.spikes_in)
mnist_clf.spikes_out.connect(output_proc.spikes_in)
# Connect Input directly to Output for ground truth labels
spike_input.label_out.connect(output_proc.label_in)
from lava.magma.core.run_conditions import RunSteps
from lava.magma.core.run_configs import Loihi1SimCfg

# Loop over all images
for img_id in range(num_images):
    print(f"\rCurrent image: {img_id+1}", end="")
```

```
    # Run each image-inference for fixed number of steps
    mnist_clf.run(
        condition=RunSteps(num_steps=num_steps_per_image),
        run_cfg=Loihi1SimCfg(select_sub_proc_model=True,
                             select_tag='fixed_pt'))

    # Reset internal neural state of LIF neurons
    mnist_clf.lif1_u.set(np.zeros((64,), dtype=np.int32))
    mnist_clf.lif1_v.set(np.zeros((64,), dtype=np.int32))
    mnist_clf.lif2_u.set(np.zeros((64,), dtype=np.int32))
    mnist_clf.lif2_v.set(np.zeros((64,), dtype=np.int32))
    mnist_clf.oplif_u.set(np.zeros((10,), dtype=np.int32))
    mnist_clf.oplif_v.set(np.zeros((10,), dtype=np.int32))

# Gather ground truth and predictions before stopping exec
ground_truth = output_proc.gt_labels.get().astype(np.int32)
predictions = output_proc.pred_labels.get().astype(np.int32)

# Stop the execution
mnist_clf.stop()

accuracy = np.sum(ground_truth==predictions)/ground_truth.size * 100

print(f"\nGround truth: {ground_truth}\n"
      f"Predictions : {predictions}\n"
      f"Accuracy    : {accuracy}")
```

In [ ]:

In [ ]:
```
import os
import numpy as np
import typing as ty

# Import Process level primitives
from lava.magma.core.process.process import AbstractProcess
from lava.magma.core.process.variable import Var
from lava.magma.core.process.ports.ports import InPort, OutPort

class SpikeInput(AbstractProcess):
    """Reads image data from the MNIST dataset and converts it to spikes.
    The resulting spike rate is proportional to the pixel value."""

    def __init__(self,
                 vth: int,
                 num_images: ty.Optional[int] = 100, #change this to be 25,50,100
                 num_steps_per_image: ty.Optional[int] = 128):
        super().__init__()
        shape = (784,)
        self.spikes_out = OutPort(shape=shape)  # Input spikes to the classifier
        self.label_out = OutPort(shape=(1,))  # Ground truth labels to OutputProc
        self.num_images = Var(shape=(1,), init=num_images)
        self.num_steps_per_image = Var(shape=(1,), init=num_steps_per_image)
        self.input_img = Var(shape=shape)
        self.ground_truth_label = Var(shape=(1,))
```

```
        self.v = Var(shape=shape, init=0)
        self.vth = Var(shape=(1,), init=vth)


class ImageClassifier(AbstractProcess):
    """A 3 layer feed-forward network with LIF and Dense Processes."""

    def __init__(self, trained_weights_path: str):
        super().__init__()

        # Using pre-trained weights and biases
        real_path_trained_wgts = os.path.realpath(trained_weights_path)

        wb_list = np.load(real_path_trained_wgts, encoding='latin1', allow_pickle=T
        w0 = wb_list[0].transpose().astype(np.int32)
        w1 = wb_list[2].transpose().astype(np.int32)
        w2 = wb_list[4].transpose().astype(np.int32)
        b1 = wb_list[1].astype(np.int32)
        b2 = wb_list[3].astype(np.int32)
        b3 = wb_list[5].astype(np.int32)

        self.spikes_in = InPort(shape=(w0.shape[1],))
        self.spikes_out = OutPort(shape=(w2.shape[0],))
        self.w_dense0 = Var(shape=w0.shape, init=w0)
        self.b_lif1 = Var(shape=(w0.shape[0],), init=b1)
        self.w_dense1 = Var(shape=w1.shape, init=w1)
        self.b_lif2 = Var(shape=(w1.shape[0],), init=b2)
        self.w_dense2 = Var(shape=w2.shape, init=w2)
        self.b_output_lif = Var(shape=(w2.shape[0],), init=b3)

        # Up-level currents and voltages of LIF Processes
        # for resetting (see at the end of the tutorial)
        self.lif1_u = Var(shape=(w0.shape[0],), init=0)
        self.lif1_v = Var(shape=(w0.shape[0],), init=0)
        self.lif2_u = Var(shape=(w1.shape[0],), init=0)
        self.lif2_v = Var(shape=(w1.shape[0],), init=0)
        self.oplif_u = Var(shape=(w2.shape[0],), init=0)
        self.oplif_v = Var(shape=(w2.shape[0],), init=0)


class OutputProcess(AbstractProcess):
    """Process to gather spikes from 10 output LIF neurons and interpret the
    highest spiking rate as the classifier output"""

    def __init__(self, **kwargs):
        super().__init__()
        shape = (10,)
        n_img = kwargs.pop('num_images', 100)
        self.num_images = Var(shape=(1,), init=n_img)
        self.spikes_in = InPort(shape=shape)
        self.label_in = InPort(shape=(1,))
        self.spikes_accum = Var(shape=shape)  # Accumulated spikes for classificati
        self.num_steps_per_image = Var(shape=(1,), init=128)
        self.pred_labels = Var(shape=(n_img,))
        self.gt_labels = Var(shape=(n_img,))
```

```
# Import parent classes for ProcessModels
from lava.magma.core.model.sub.model import AbstractSubProcessModel
from lava.magma.core.model.py.model import PyLoihiProcessModel

# Import ProcessModel ports, data-types
from lava.magma.core.model.py.ports import PyInPort, PyOutPort
from lava.magma.core.model.py.type import LavaPyType

# Import execution protocol and hardware resources
from lava.magma.core.sync.protocols.loihi_protocol import LoihiProtocol
from lava.magma.core.resources import CPU

# Import decorators
from lava.magma.core.decorator import implements, requires

# Import MNIST dataset
from lava.utils.dataloader.mnist import MnistDataset
np.set_printoptions(linewidth=np.inf)

@implements(proc=SpikeInput, protocol=LoihiProtocol)
@requires(CPU)
class PySpikeInputModel(PyLoihiProcessModel):
    num_images: int = LavaPyType(int, int, precision=32)
    spikes_out: PyOutPort = LavaPyType(PyOutPort.VEC_DENSE, bool, precision=1)
    label_out: PyOutPort = LavaPyType(PyOutPort.VEC_DENSE, np.int32,
                                      precision=32)
    num_steps_per_image: int = LavaPyType(int, int, precision=32)
    input_img: np.ndarray = LavaPyType(np.ndarray, int, precision=32)
    ground_truth_label: int = LavaPyType(int, int, precision=32)
    v: np.ndarray = LavaPyType(np.ndarray, int, precision=32)
    vth: int = LavaPyType(int, int, precision=32)

    def __init__(self, proc_params):
        super().__init__(proc_params=proc_params)
        self.mnist_dataset = MnistDataset()
        self.curr_img_id = 0

    def post_guard(self):
        """Guard function for PostManagement phase.
        """
        if self.time_step % self.num_steps_per_image == 1:
            return True
        return False

    def run_post_mgmt(self):
        """Post-Management phase: executed only when guard function above
        returns True.
        """
        img = self.mnist_dataset.images[self.curr_img_id]
        self.ground_truth_label = self.mnist_dataset.labels[self.curr_img_id]
        self.input_img = img.astype(np.int32) - 127
        self.v = np.zeros(self.v.shape)
        self.label_out.send(np.array([self.ground_truth_label]))
        self.curr_img_id += 1

    def run_spk(self):
```

```
        """Spiking phase: executed unconditionally at every time-step
        """
        self.v[:] = self.v + self.input_img
        s_out = self.v > self.vth
        self.v[s_out] = 0  # reset voltage to 0 after a spike
        self.spikes_out.send(s_out)

from lava.proc.lif.process import LIF
from lava.proc.dense.process import Dense

@implements(ImageClassifier)
@requires(CPU)
class PyImageClassifierModel(AbstractSubProcessModel):
    def __init__(self, proc):
        self.dense0 = Dense(weights=proc.w_dense0.init)
        self.lif1 = LIF(shape=(64,), bias_mant=proc.b_lif1.init, vth=400,
                        dv=0, du=4095)
        self.dense1 = Dense(weights=proc.w_dense1.init)
        self.lif2 = LIF(shape=(64,), bias_mant=proc.b_lif2.init, vth=350,
                        dv=0, du=4095)
        self.dense2 = Dense(weights=proc.w_dense2.init)
        self.output_lif = LIF(shape=(10,), bias_mant=proc.b_output_lif.init,
                              vth=1, dv=0, du=4095)

        proc.spikes_in.connect(self.dense0.s_in)
        self.dense0.a_out.connect(self.lif1.a_in)
        self.lif1.s_out.connect(self.dense1.s_in)
        self.dense1.a_out.connect(self.lif2.a_in)
        self.lif2.s_out.connect(self.dense2.s_in)
        self.dense2.a_out.connect(self.output_lif.a_in)
        self.output_lif.s_out.connect(proc.spikes_out)

        # Create aliases of SubProcess variables
        proc.lif1_u.alias(self.lif1.u)
        proc.lif1_v.alias(self.lif1.v)
        proc.lif2_u.alias(self.lif2.u)
        proc.lif2_v.alias(self.lif2.v)
        proc.oplif_u.alias(self.output_lif.u)
        proc.oplif_v.alias(self.output_lif.v)

num_images = 25 #change this to be 25,50,100
num_steps_per_image = 128

# Create Process instances
spike_input = SpikeInput(vth=1,
                         num_images=num_images,
                         num_steps_per_image=num_steps_per_image)
mnist_clf = ImageClassifier(
    trained_weights_path=os.path.join('.', 'mnist_pretrained.npy'))
output_proc = OutputProcess(num_images=num_images)

# Connect Processes
spike_input.spikes_out.connect(mnist_clf.spikes_in)
mnist_clf.spikes_out.connect(output_proc.spikes_in)
# Connect Input directly to Output for ground truth labels
spike_input.label_out.connect(output_proc.label_in)
```

```
from lava.magma.core.run_conditions import RunSteps
from lava.magma.core.run_configs import Loihi1SimCfg

# Loop over all images
for img_id in range(num_images):
    print(f"\rCurrent image: {img_id+1}", end="")

    # Run each image-inference for fixed number of steps
    mnist_clf.run(
        condition=RunSteps(num_steps=num_steps_per_image),
        run_cfg=Loihi1SimCfg(select_sub_proc_model=True,
                             select_tag='fixed_pt'))

    # Reset internal neural state of LIF neurons
    mnist_clf.lif1_u.set(np.zeros((64,), dtype=np.int32))
    mnist_clf.lif1_v.set(np.zeros((64,), dtype=np.int32))
    mnist_clf.lif2_u.set(np.zeros((64,), dtype=np.int32))
    mnist_clf.lif2_v.set(np.zeros((64,), dtype=np.int32))
    mnist_clf.oplif_u.set(np.zeros((10,), dtype=np.int32))
    mnist_clf.oplif_v.set(np.zeros((10,), dtype=np.int32))

# Gather ground truth and predictions before stopping exec
ground_truth = output_proc.gt_labels.get().astype(np.int32)
predictions = output_proc.pred_labels.get().astype(np.int32)

# Stop the execution
mnist_clf.stop()

accuracy = np.sum(ground_truth==predictions)/ground_truth.size * 100

print(f"\nGround truth: {ground_truth}\n"
      f"Predictions : {predictions}\n"
      f"Accuracy    : {accuracy}")
```