Spike-based machine learning with GeNN

James Knight and Thomas Nowotny
J.C.Knight@sussex.ac.uk

brains on BOARD

University of Sussex

GeNN

- Cross-platform C++ library for generating optimised CUDA code for GPU accelerated SNN simulations.
- Can also generate C++ code for testing on computers without GPUs (e.g. here with your laptops!)
- Hopefully you've learnt all about it in our talks earlier in the week!
- All GeNN features are now available from Python for easier interoperability with other ML and Computational Neuroscience tools

Installation

CUDA on Linux

- Each version of CUDA only supports a subset of GCC versions so if you have a very old or very bleeding edge OS you may need to install an additional version of GCC.
- Installing CUDA via the NVIDIA proprietary packages tends to work best if your OS is supported.
- Ensure that the CUDA_PATH environment variable is set

CUDA on Windows

- CUDA is nicely integrated into Visual Studio and provided graphical debugging and profiling tools
- Historically, because Visual Studio is frequently updated, compiler/CUDA version mismatches were more prevalent than on Linux **but**, as of CUDA 10 and Visual Studio 2017, this no longer appears to be the case!
- If installing from scratch we recommend:
    - CUDA 10.1
    - Visual Studio 2017

CUDA on Mac

- Sadly Apple hasn't built any machines with NVIDIA GPUs since 2014
- However, if you're lucky enough to have:
  - MacBook Pro (Retina, 15-inch, Late 2013)
  - MacBook Pro (Retina, 15-inch, Mid 2014)
  - Equivalent iMac models (probably not with you!)
- You **may** have a NVIDIA GPU that's usable with the current version of CUDA!
- Ensure that the CUDA_PATH environment variable is set

Windows notes

- PyGeNN requires at least Python 3.5 and Visual Studio 2015 installed
- I recommend using Anaconda for Python development on Windows
- You should install CUDA **after** Visual Studio so they are correctly integrated
- PyGeNN should be used from a command prompt with Python **and** Visual studio properly configured i.e. by activating conda within a Visual Studio "x64 Native Tools Command Prompt"

Installing PyGeNN from binary wheels

1. Select a suitable wheel from the latest release available at
   https://github.com/genn-team/genn/releases
   For example, if you have a Linux system with Python 3.7, you would pick `pygenn-0.2-cp37-cp37m-linux_x86_64.whl`
   Note: the Mac OS X wheel are built for CUDA 9, all others for CUDA 10
2. Install the wheel using pip e.g.
   `pip install pygenn-0.2-cp37-cp37m-linux_x86_64.whl`

Installation from source on Linux/Mac

1. Download latest release of GeNN from
   https://github.com/genn-team/genn/releases
2. Make sure you have swig installed
3. From GeNN directory, build as a dynamic library, directly into the PyGeNN directory using:
   ```
   make DYNAMIC=1
   LIBRARY_DIRECTORY=`pwd`/pygenn/genn_wrapper/
   ```
4. Build and install python module with setuptools using:
   ```
   python setup.py develop
   ```

Installation from source on Windows

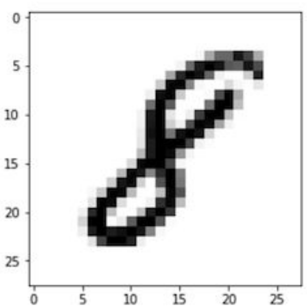
1. Download latest release of GeNN from
   https://github.com/genn-team/genn/releases
2. Make sure you have swig installed
3. From GeNN directory, build as a dynamic library using:
   `msbuild genn.sln /t:Build /p:Configuration=Release_DLL`
4. Copy the newly built DLLs into pygenn using
   `copy /Y lib\genn*Release_DLL.* pygenn\genn_wrapper`
5. Build and install python module with setuptools using:
   `python setup.py develop`

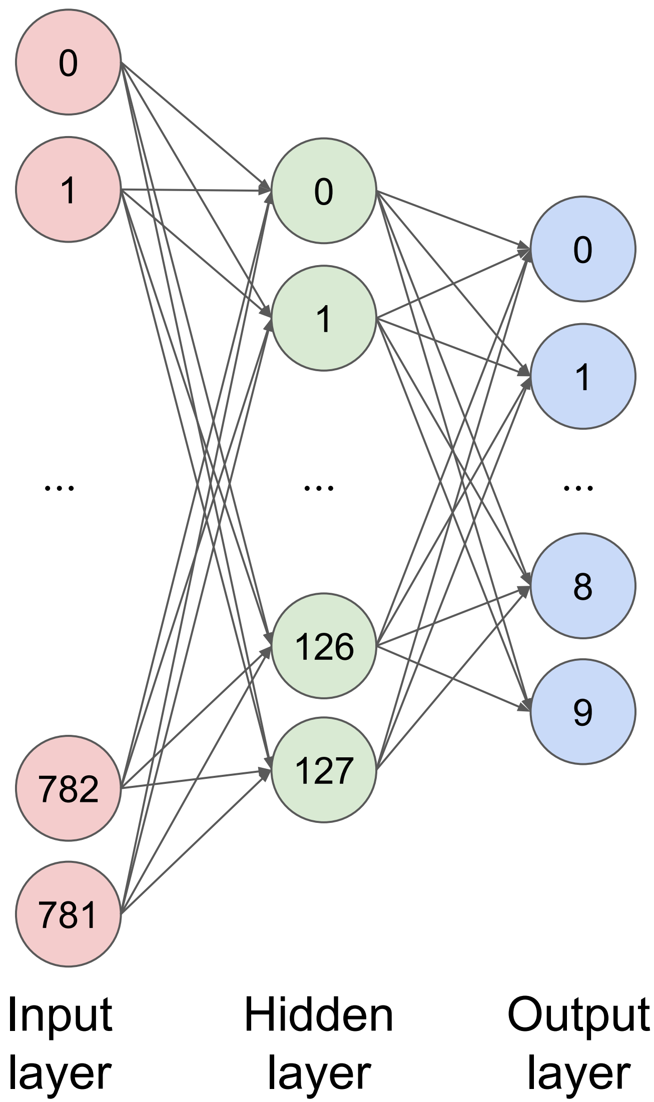
Tutorial repository

- Checkout from:
  https://github.com/neworderofjamie/pygenn_ml_tutorial
- Contains
  - testing_images.npy and testing_labels.npy - testing portion of MNIST dataset
  - weights_0_1.npy and weights_1_0.npy - weights trained in Keras
  - tutorial_1.py and tutorial_2.py - Code for this tutorial
- Test your installation with:
  python tutorial_1.py

Tutorial: MNIST inference

Part 0: Training the ANN





- Not state-of-the-art!
- I have already done this part for you!
- Achieves 97.6% accuracy on MNIST

Part 1: Classifying a single image

1. Basics of using PyGeNN
2. Building a spiking network based on ANN
3. Recording spikes resulting from presenting single MNIST digit

```python
import numpy as np
from os import path

from pygenn.genn_model import (create_custom_neuron_class,
                               create_custom_current_source_class,
                               GeNNModel)
from pygenn.genn_wrapper import NO_DELAY
```

Import some standard Python packages

Import some useful PyGeNN components

```python
import numpy as np
from os import path

from pygenn.genn_model import (create_custom_neuron_class,
                               create_custom_current_source_class,
                               GeNNModel)
from pygenn.genn_wrapper import NO_DELAY

# ----------------------------------------------------------------------------
# Parameters
# ----------------------------------------------------------------------------
IF_PARAMS = {"Vthr": 5.0}
TIMESTEP = 1.0
PRESENT_TIMESTEPS = 100
INPUT_CURRENT_SCALE = 1.0 / 100.0
```

Input required for neurons to spike

How long to present each digit

How to scale image intensity to input currents

```python
# ----------------------------------------------------------------------------
# Custom GeNN models
# ----------------------------------------------------------------------------
# Very simple integrate-and-fire neuron model
if_model = create_custom_neuron_class(
    "if_model",
    param_names=["Vthr"],
    var_name_types=[("V", "scalar"), ("SpikeCount", "unsigned int")],
    sim_code="$(V) += $(Isyn) * DT;",
    reset_code="""
    $(V) = 0.0;
    $(SpikeCount)++;
    """,
    threshold_condition_code="$(V) >= $(Vthr)")
```

Internal name of model - must be unique

```python
# ----------------------------------------------------------------------------
# Custom GeNN models
# ----------------------------------------------------------------------------
# Very simple integrate-and-fire neuron model
if_model = create_custom_neuron_class(
    "if_model",
    param_names=["Vthr"],
    var_name_types=[("V", "scalar"), ("SpikeCount", "unsigned int")],
    sim_code="$(V) += $(Isyn) * DT;",
    reset_code="""
    $(V) = 0.0;
    $(SpikeCount)++;
    """,
    threshold_condition_code="$(V) >= $(Vthr)
```

Parameters are common across all neurons in population (layer)

State variables used to track per-neuron membrane voltage and spike count

```python
# ------------------------------------------------------------------------------
# Custom GeNN models
# ------------------------------------------------------------------------------
# Very simple integrate-and-fire neuron model
if_model = create_custom_neuron_class(
    "if_model",
    param_names=["Vthr"],
    var_name_types=[("V", "scalar"), ("SpikeCount", "unsigned int")],
    sim_code="$(V) += $(Isyn) * DT;",
    reset_code="""
    $(V) = 0.0;
    $(SpikeCount)++;
    """,
    threshold_condition_code="$(V) >= $(Vthr)")
```

Parameters are common across all neurons in population (layer)

Neuron simply integrates input current $(Isyn) every timestep

```
# ---------------------------------------------------------------------------
# Custom GeNN models
# ---------------------------------------------------------------------------
# Very simple integrate-and-fire neuron model
if_model = create_custom_neuron_class(
    "if_model",
    param_names=["Vthr"],
    var_name_types=[("V", "scalar"), ("SpikeCount", "unsigned int")],
    sim_code="$(V) += $(Isyn) * DT;",
    reset_code="""
    $(V) = 0.0;
    $(SpikeCount)++;
    """,
    threshold_condition_code="$(V) >= $(Vthr)")
```

When neuron spikes, $(V) is reset and $(SpikeCount) is incremented

Neuron spikes when $(V) goes above $(Vthr)

```python
# ------------------------------------------------------------------------------
# Custom GeNN models
# ------------------------------------------------------------------------------
# Very simple integrate-and-fire neuron model
if_model = create_custom_neuron_class(
    "if_model",
    param_names=["Vthr"],
    var_name_types=[("V", "scalar"), ("SpikeCount", "unsigned int")],
    sim_code="$(V) += $(Isyn) * DT;",
    reset_code="""
    $(V) = 0.0;
    $(SpikeCount)++;
    """,
    threshold_condition_code="$(V) >= $(Vthr)")

# Current source model which injects current with a magnitude specified by a state variable
cs_model = create_custom_current_source_class(
    "cs_model",
    var_name_types=[("magnitude", "scalar")],
    injection_code="$(injectCurrent, $(magnitude));")
```

Current source ‘injects’ current specified by state variable each timestep

```
# ----------------------------------------------------------------------------
# Build model
# -----------------------------------------
# Create GeNN model
model = GeNNModel("float", "tutorial_1")
model.dT = TIMESTEP

# Load weights
weights = []
while True:
    filename = "weights_%u_%u.npy" % (len(weights), len(weights) + 1)
    if path.exists(filename):
        weights.append(np.load(filename))
    else:
        break
```

Create new network using single-precision by default and generating code into tutorial_1 directory

```python
# ----------------------------------------------------------------------------
# Build model
# ----------------------------------------------------------------------------
# Create GeNN model
model = GeNNModel("float", "tutorial_1")
model.dT = TIMESTEP

# Load weights
weights = []
while True:
    filename = "weights_%u_%u.npy" % (len(weights), len(weights) + 1)
    if path.exists(filename):
        weights.append(np.load(filename))
    else:
        break
```

Set simulation time-step (somewhat arbitrary with artifical models)

Load any weights present in directory into list

```python
# ----------------------------------------------------------------------------
# Build model
# ----------------------------------------------------------------------------
# Create GeNN model
model = GeNNModel("float", "spiking_eval")
model.dT = TIMESTEP

# Load weights
weights = []
while True:
    filename = "weights_%u_%u.npy" % (len(weights), len(weights) + 1)
    if path.exists(filename):
        weights.append(np.load(filename))
    else:
        break

# Initial values to initialise all neurons f
if_init = {"V": 0.0, "SpikeCount":0}

# Create first neuron layer
neuron_layers = [model.add_neuron_population("neuron0", weights[0].shape[0],
                                             if_model, IF_PARAMS, if_init)]

# Create subsequent neuron layer
for i, w in enumerate(weights)
    neuron_layers.append(model
```

Initial values for all IF neurons

Create neuron population with a neuron for each of the first layer’s input dimensions

```python
# ----------------------------------------------------------------------------
# Build model
# ----------------------------------------------------------------------------
# Create GeNN model
model = GeNNModel("float", "spiking_eval")
model.dT = TIMESTEP

# Load weights
weights = []
while True:
    filename = "weights_%u_%u.npy" % (len(weights), len(weights) + 1)
    if path.exists(filename):
        weights.append(np.load(filename))
    else:
        break

# Initial values to initialise all neurons to
if_init = {"V": 0.0, "SpikeCount":0}

# Create first neuron layer
neuron_layers = [model.add_neuron_population("neuron0", weights[0].shape[0],
                                             if_model, IF_PARAMS, if_init)]

# Create subsequent neuron layer
for i, w in enumerate(weights):
    neuron_layers.append(model.add_neuron_population("neuron%u" % (i + 1),
                                                     w.shape[1], if_model,
                                                     IF_PARAMS, if_init))
```

Create neuron populations matching subsequent layer’s output dimensions

```python
# Create synaptic connections between layers
for i, (pre, post, w) in enumerate(zip(neuron_layers[:-1], neuron_layers[1:], weights)):
    model.add_synapse_population(
        "synapse%u" % i, "DENSE_INDIVIDUALG", NO_DELAY,
        pre, post,
        "StaticPulse", {}, {"g": w.flatten()}, {}, {},
        "DeltaCurr", {}, {})
```

No synaptic delays

Name of synapse population

Dense matrix with individual state variables (weights) for each synapse
http://genn-team.github.io/genn/documentation/4/html/d5/d39/subsect34.html

```python
# Create synaptic connections between layers
for i, (pre, post, w) in enumerate(zip(neuron_layers[:-1], neuron_layers[1:], weights)):
    model.add_synapse_population(
        "synapse%u" % i, "DENSE_INDIVIDUALG", NO_DELAY,
        pre, post,
        "StaticPulse", {}, {"g": w.flatten()}, {}, {},
        "DeltaCurr", {}, {})
```

Source population

Target population

```python
# Create synaptic connections between layers
for i, (pre, post, w) in enumerate(zip(neuron_layers[:-1], neuron_layers[1:], weights)):
    model.add_synapse_population(
        "synapse%u" % i, "DENSE_INDIVIDUALG", NO_DELAY,
        pre, post,
        "StaticPulse", {}, {"g": w.flatten()}, {}, {},
        "DeltaCurr", {}, {})
```

Static synapses have no parameters

Use built-in static synapse model

Initialise the weight of each synapse to the pre-trained weights

```python
# Create synaptic connections between layers
for i, (pre, post, w) in enumerate(zip(neuron_layers[:-1], neuron_layers[1:], weights)):
    model.add_synapse_population(
        "synapse%u" % i, "DENSE_INDIVIDUALG", NO_DELAY,
        pre, post,
        "StaticPulse", {}, {"g": w.flatten()}, {}, {},
        "DeltaCurr", {}, {})
```

Use built-in delta postsynaptic model

This model has no parameters or variables

```
# Create synaptic connections between layers
for i, (pre, post, w) in enumerate(zip(neuron_layers[:-1], neuron_layers[1:], weights)):
    model.add_synapse_population(
        "synapse%u" % i, "DENSE_INDIVI
        pre, post,
        "StaticPulse", {}, {"g": w.fla
        "DeltaCurr", {}, {})

# Create current source to deliver input to first layers of neurons
current_input = model.add_current_source("current_input", cs_model,
                                         "neuron0" , {}, {"magnitude": 0.0})

# Build and load our model
model.build()
model.load()
```

Add current source to provide input into first layer (neuron0)

Uses GeNN to generate simulation code for model

Loads generated model into PyGeNN

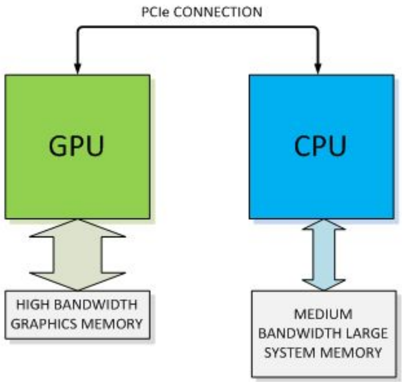
```python
# ----------------------------------------------------------------------------
# Simulate
# ----------------------------------------------------------------------------
# Load testing data
testing_images = np.load("testing_images.npy")
testing_labels = np.load("testing_labels.npy")

# Check dimensions match network
assert testing_images.shape[1] == weights[0].shape[0]
assert np.max(testing_labels) == (weights[1].shape[1] - 1)

# Set current input by scaling first image
current_input.vars["magnitude"].view[:] = testing_images[0] * INPUT_CURRENT_SCALE
```

Load MNIST data

Copy first image into memory view of current source magnitude

```python
# ---------------------------------------------------------------------------
# Simulate
# ---------------------------------------------------------------------------
# Load testing data
testing_images = np.load("testing_images.npy")
testing_labels = np.load("testing_labels.npy")

# Check dimensions match network
assert testing_images.shape[1] == weights[0].shape[0]
assert np.max(testing_labels) == (weights[1].shape[1] - 1)

# Set current input by scaling first image
current_input.vars["magnitude"].view[:] = testing_images[0] * INPUT_CURRENT_SCALE

# Upload
model.push_var_to_device("current_input", "magnitude")
```

Upload this variable to GPU



```
# Simulate
layer_spikes = [(np.empty(0), np.empty(0)) for _ in enumerate(neuron_layers)]
while model.timestep < PRESENT_TIMESTEPS:
    # Advance simulation
    model.step_time()

    # Loop through neuron layers
    for i, l in enumerate(neuron_layers):
        # Download spikes
        model.pull_current_spikes_from_device(l.name)

        # Add to data structure
        spike_times = np.ones_like(l.current_spikes) * model.t
        layer_spikes[i] = (np.hstack((layer_spikes[i][0], l.current_spikes)),
                           np.hstack((layer_spikes[i][1], spike_times)))
```

Advance simulation

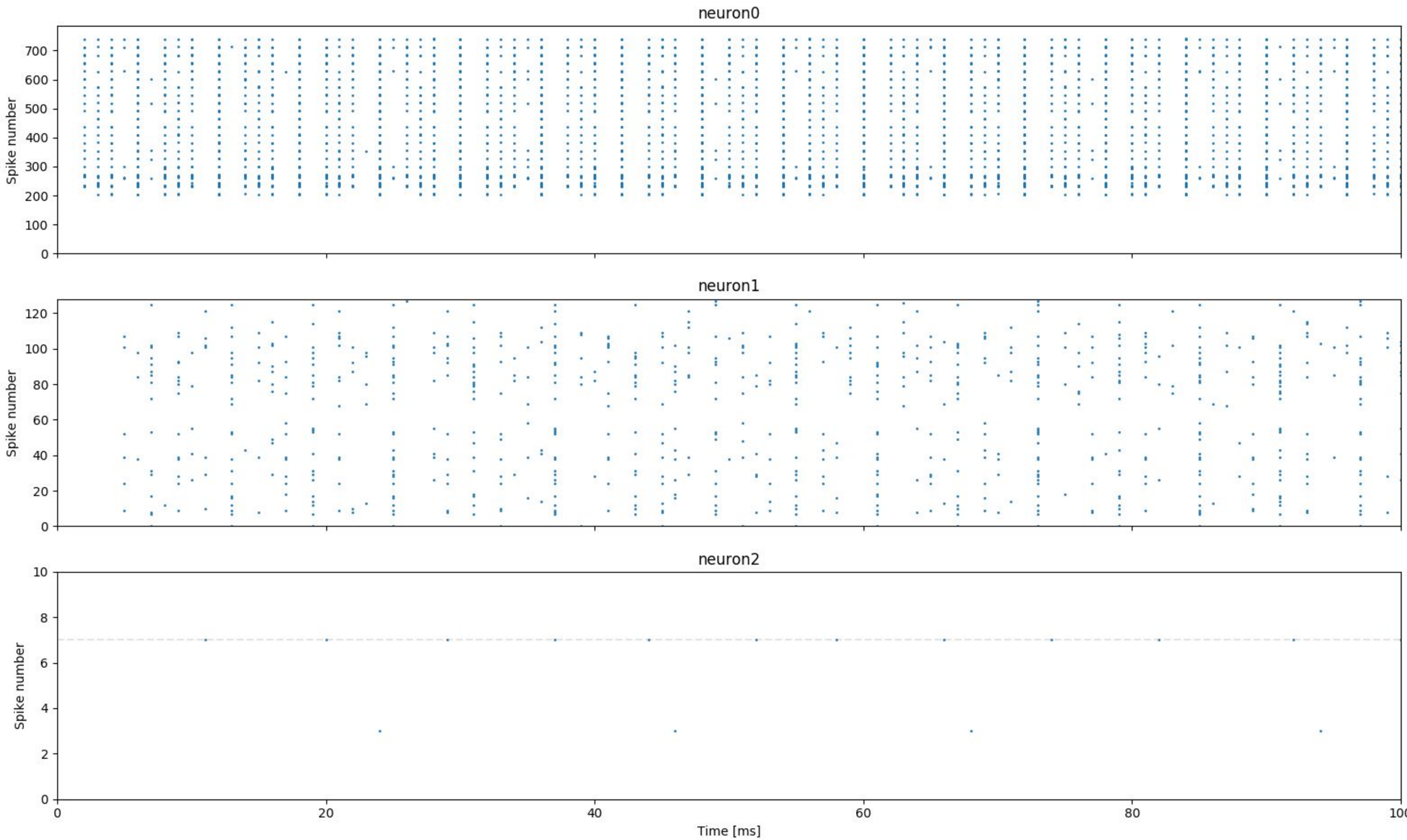
Download spikes emitted this timestep from GPU

Add spikes to data structure

Time [ms]	0	0	1	1	2
Neuron index	1	14	13	100	56

```python
# -------------------------------------------------------------------------------
# Plotting
# -------------------------------------------------------------------------------
import matplotlib.pyplot as plt

# Create a plot with axes for each
fig, axes = plt.subplots(len(neuron_layers), sharex=True)

# Loop through axes and their corresponding neuron populations
for a, s, l in zip(axes, layer_spikes, neuron_layers):
    # Plot spikes
    a.scatter(s[1], s[0], s=1)

    # Set title, axis labels
    a.set_title(l.name)
    a.set_ylabel("Spike number")
    a.set_xlim((0, PRESENT_TIMESTEPS * TIMESTEP))
    a.set_ylim((0, l.size))

# Add an x-axis label and translucent line showing the correct label
axes[-1].set_xlabel("Time [ms]")
axes[-1].hlines(testing_labels[0], xmin=0, xmax=PRESENT_TIMESTEPS,
                linestyle="--", color="gray", alpha=0.2)

# Show plot
plt.show()
```

Results



neuron0
Spike number
700
600
500
400
300
200
100
0
neuron1
Spike number
120
100
80
60
40
20
0
neuron2
Spike number
10
8
6
4
2
0
0
20
40
60
80
100
Time [ms]

Part 2: Evaluating entire dataset

1. Presenting entire MNIST testing set to network
2. Calculating inference performance

```python
# ----------------------------------------------------------------------------
# Simulate
# ----------------------------------------------------------------------------
# Load testing data
testing_images = np.load("testing_images.npy")
testing_labels = np.load("testing_labels.npy")

# Check dimensions match network
assert testing_images.shape[1] == weights[0].shape[0]
assert np.max(testing_labels) == (weights[1].shape[1] - 1)

# Get views to efficiently access state variables
current_input_magnitude = current_input.vars["magnitude"].view
output_spike_count = neuron_layers[-1].vars["SpikeCount"].view
layer_voltages = [l.vars["V"].view for l in neuron_layers]
```

Cache memory views of required state variables

```python
# Simulate
num_correct = 0
while model.timestep < (PRESENT_TIMESTEPS * testing_images.shape[0]):
    # Calculate the timestep within the presentation
    timestep_in_example = model.timestep % PRESENT_TIMESTEPS
    example = int(model.timestep // PRESENT_TIMESTEPS)

    # If this is the first timestep of presenting the example
    if timestep_in_example == 0:
        current_input_magnitude[:] = testing_images[example]
        model.push_var_to_device("current_input", "magnitude")

        # Loop through all layers and their corresponding voltage views
        for l, v in zip(neuron_layers, layer_voltages):
            # Manually 'reset' voltage
            v[:] = 0.0

            # Upload
            model.push_var_to_device(l.name, "V")

        # Zero spike count
        output_spike_count[:] = 0
        model.push_var_to_device(neuron_layers[-1].name, "SpikeCount")

    # Advance simulation
    model.step_time()
```

Divide timestep into example index and timestep within example

```python
# Simulate
num_correct = 0
while model.timestep < (PRESENT_TIMESTEPS * testing_images.shape[0]):
    # Calculate the timestep within the presentati
    timestep_in_example = model.timestep % PRESENT_
    example = int(model.timestep // PRESENT_TIMEST

    # If this is the first timestep of presenting the example
    if timestep_in_example == 0:
        current_input_magnitude[:] = testing_images[example] * INPUT_CURRENT_SCALE
        model.push_var_to_device("current_input", "magnitude")

        # Loop through all layers and their corresponding voltage views
        for l, v in zip(neuron_layers, layer_voltages):
            # Manually 'reset' voltage
            v[:] = 0.0

            # Upload
            model.push_var_to_device(l.name, "V")

        # Zero spike count
        output_spike_count[:] = 0
        model.push_var_to_device(neuron_layers[-1].name, "SpikeCount")

    # Advance simulation
    model.step_time()
```

Copy image into memory view and upload to GPU

Set all neuron voltages to zero

Upload to GPU

```python
# Simulate
num_correct = 0
while model.timestep < (PRESENT_TIMESTEPS * testing_images.shape[0]):
    # Calculate the timestep within the presentation
    timestep_in_example = model.timestep % PRESENT_TIMESTEPS
    example = int(model.timestep // PRESENT_TIMESTEPS)

    # If this is the first timestep of presenting the example
    if timestep_in_example == 0:
        current_input_magnitude[:] = testing_images[example] * INPUT_CURRENT_SCALE
        model.push_var_to_device("current_input", "magnitude")

        # Loop through all layers and their corresponding voltage views
        for l, v in zip(neuron_layers, layer_voltages):
            # Manually 'reset' voltage
            v[:] = 0.0

            # Upload
            model.push_var_to_device(l.name, "V")

        # Zero spike count
        output_spike_count[:] = 0
        model.push_var_to_device(neuron_layers[-1].name, "SpikeCount")

    # Advance simulation
    model.step_time()
```

Zero spike count for all output layer neurons and upload to GPU

Advance simulation

```python
    # If this is the LAST timestep of presenting the example
    if timestep_in_example == (PRESENT_TIMESTEPS - 1):
        # Download spike count from last layer
        model.pull_var_from_device(neuron_layers[-1].name, "SpikeCount")

        # Find which neuron spiked the most to get prediction
        predicted_label = np.argmax(output_spike_count)
        true_label = testing_labels[example]

        print("\tExample=%u, true label=%u, predicted label=%u" % (example,
                                                                  true_label,
                                                                  predicted_label))


        if predicted_label == true_label:
            num_correct += 1

print("Accuracy %f%%" % ((num_correct / float(testing_images.shape[0])) * 100.0))
```

Download output layer spike count

Results

```
        Example=9965, true label=3, predicted label=3
        Example=9966, true label=6, predicted label=6
        Example=9967, true label=8, predicted label=8
        Example=9968, true label=7, predicted label=7
        Example=9969, true label=1, predicted label=1
        Example=9970, true label=5, predicted label=5
        Example=9971, true label=2, predicted label=2
        Example=9972, true label=4, predicted label=4
        Example=9973, true label=9, predicted label=9
        Example=9974, true label=4, predicted label=4
        Example=9975, true label=3, predicted label=3
        Example=9976, true label=6, predicted label=6
        Example=9977, true label=4, predicted label=4
        Example=9978, true label=1, predicted label=1
        Example=9979, true label=7, predicted label=7
        Example=9980, true label=2, predicted label=2
        Example=9981, true label=6, predicted label=6
        Example=9982, true label=5, predicted label=5
        Example=9983, true label=0, predicted label=0
        Example=9984, true label=1, predicted label=1
        Example=9985, true label=2, predicted label=2
        Example=9986, true label=3, predicted label=3
        Example=9987, true label=4, predicted label=4
        Example=9988, true label=5, predicted label=5
        Example=9989, true label=6, predicted label=6
        Example=9990, true label=7, predicted label=7
        Example=9991, true label=8, predicted label=8
        Example=9992, true label=9, predicted label=9
        Example=9993, true label=0, predicted label=0
        Example=9994, true label=1, predicted label=1
        Example=9995, true label=2, predicted label=2
        Example=9996, true label=3, predicted label=3
        Example=9997, true label=4, predicted label=4
        Example=9998, true label=5, predicted label=5
        Example=9999, true label=6, predicted label=6
Accuracy 97.440000%
(tensorflow) jk421@inf900801:~/offline_train_example$
```

Part 3: Play time!

- How does PRESENT_TIMESTEPS affect performance?
- Can you reduce the number of spikes while maintaining performance by modifying IF_PARAMS and INPUT_CURRENT_SCALE?
- Try training your own sequential model with dense layers, save the weights and see how this code performs

Thank you!

J.C.Knight@sussex.ac.uk