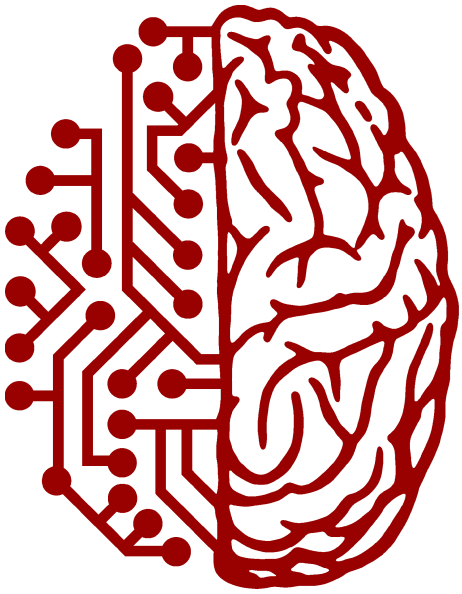
PyNN - Code Collection

Für die Benutzung von Spikey



Andreas Baumbach & Dominik Dold

5.07.2017

PyNN - Code Collection

Andreas Baumbach Dominik Dold

5.07.2017

Inhaltsverzeichnis

1 Vorwort **4**

2 PyNN-Command Sammlung **4**
2.1 Grundbefehle . . . 4
2.1.1 pynn.setup . . . 4
2.1.2 pynn.Population . . . 4
2.1.3 pynn.Projection . . . 4
2.1.4 pynn.run . . . 5
2.1.5 pynn.end . . . 5
2.2 Neuronen . . . 5
2.2.1 pynn.IF_facets_hardware1 . . . 5
2.2.2 pynn.IF_facets_hardware1.default_parameters . . . 5
2.3 Connectors . . . 5
2.3.1 pynn.AllToAllConnector . . . 5
2.3.2 pynn.FixedNumberPostConnector . . . 6
2.3.3 pynn.FixedNumberPreConnector . . . 6
2.3.4 pynn.FromListConnector . . . 6
2.3.5 pynn.OneToOneConnector . . . 6
2.4 Sources . . . 7
2.4.1 pynn.SpikeSourceArray . . . 7
2.4.2 pynn.SpikeSourcePoisson . . . 7
2.5 Readout . . . 7
2.5.1 pynn.record . . . 7
2.5.2 pynn.getSpikes . . . 7
2.5.3 pynn.record_v . . . 8
2.5.4 pynn.timeMembraneOutput . . . 8
2.5.5 pynn.membraneOutput . . . 8
2.6 Hardware Gewichte . . . 8
2.6.1 pynn.maxExcWeight . . . 8
2.6.2 pynn.maxInhWeight . . . 8
2.6.3 pynn.minExcWeight . . . 9
2.6.4 pynn.minInhWeight . . . 9
2.7 Minimales Beispiel . . . 9

1 Vorwort

Auf den folgenden Seiten werden die für die Benutzung von Spikey wichtigsten PyNN-Befehle vorgestellt und anhand Syntax-Beispielen erklärt. Zusätzlich befindet sich am Ende des Dokuments ein minimales Code-Beispiel für die typische Verwendung von PyNN. Bitte beachten Sie, dass dieses Dokument nicht alle PyNN-Befehle abdeckt. Weitere Informationen zu PyNN können auf `http://neuralensemble.org/docs/PyNN/` gefunden werden. Für die Darstellung von Python Code wurde der Python Highlighting Style von Olivier Verdier verwendet (`https://github.com/olivierverdier/python-latex-highlighting`).

2 PyNN-Command Sammlung

2.1 Grundbefehle

Grundlegende Befehle die notwendig sind für jede Simulation.

2.1.1 pynn.setup

Wird zu Beginn jedes Experiments aufgerufen.

```
pynn.setup(mappingoffset = 2)
```

Der Parameter *mappingoffset* gibt an, bei welchem physikalischen Neuron auf dem Spikey Chip das Mapping beginnt.

2.1.2 pynn.Population

Erstellt eine Population (Gruppe) aus Neuronen.

```
pynn.Population(dims=5, cellclass = pynn.IF_facets_hardware1,
                cellparams = params, label = '')
```

Der Parameter *dims* gibt die Anzahl Neuronen in der Population an. Die Neuronen werden sequenziell auf Spikey platziert beginnend bei dem Neuron mit ID *mappingoffset* (siehe pynn.setup). Weitere Populationen werden entsprechend hinter der zuletzt erstellten Population platziert.
cellclass gibt das Neuronenmodell an, in unserem Falle Spikey Neuronen (*pynn.IF_facets_hardware1*).
cellparams beinhaltet die konfigurierbaren Parameter des Neuronenmodells, für das Format siehe *pynn.IF_facets_hardware1.default_parameters*.

2.1.3 pynn.Projection

Erzeugt Verbindungen zwischen Populationen von Neuronen.

```
pynn.Projection(presynaptic_population = pop1,
                postsynaptic_population = pop2,
                method = connector, target = 'excitatory')
```

Die *postsynaptic_population* empfängt die Spikes der *presynaptic_population*.
method gibt an, wie einzelne Neuronen zwischen den Populationen verbunden werden (siehe *Connectors*).
target bestimmt den Typ der Interaktion (excitatory - anregend, inhibitory - unterdrückend).

2.1.4 pynn.run

Startet die Simulation für das spezifizierte Netzwerk (gegeben durch Populationen und Projektionen).

```
pynn.run(simtime=1000.)
```

Führt die Simulation für *simtime* Millisekunden aus.

2.1.5 pynn.end

Beendet die Simulation. Notwendig bevor nochmals *pynn.setup* aufgerufen werden kann. Nötig um mehrere Simulationen in einem Skript zu starten.

2.2 Neuronen

2.2.1 pynn.IF_facets_hardware1

Neurontyp des Spikey Chips.

```
pynn.IF_facets_hardware1(parameters = params)
```

parameters sind die Neuronen-Parameter die im Hardware-Neuron eingestellt werden sollen. Die Parameter werden durch ein dictionary eingestellt. Wird keines spezifiziert, werden die folgenden Standardwerte benutzt:

2.2.2 pynn.IF_facets_hardware1.default_parameters

```
pynn.IF_facets_hardware1.default_parameters = {
    "e_rev_I": -80.0,
    "g_leak": 20.0,
    "tau_refrac": 1.0,
    "v_reset": -80.0,
    "v_rest": -75.0,
    "v_thresh": -55.0
}
```

e_rev_I setzt das inhibitorische Umkehrpotential in Millivolt.
g_leak setzt die Leck-Konduktanz zum Ruhepotential *V_rest*
tau_refrac setzt die Refraktärzeit des Neurons in Millisekunden.
v_reset setzt das Resetpotential in Millivolt.
v_rest setzt das Ruhepotential in Millivolt.
v_thresh setzt die Feuerschwelle in Millivolt.

2.3 Connectors

Connectors implementieren Verbindungsarten zwischen Populationen.

2.3.1 pynn.AllToAllConnector

Verbindet alle Neuronen der presynaptischen Population mit allen Neuronen der postsynaptischen Population einer Projektion (siehe *pynn.Projection*).

```
pynn.AllToAllConnector(allow_self_connections = True, weights = 1.0)
```

Wenn *allow_self_connections* auf True gesetzt ist (anstatt False), werden einzelne Neuronen auch mit sich selbst verbunden. Dies greift nur, wenn die pre- und postsynaptische Populationen identisch sind (d.h. eine Population mit sich selbst verbunden wird).
weights gibt die Stärke der Verbindungen in nS (Nanosiemens) an (siehe *Hardware Gewichte*).

2.3.2 pynn.FixedNumberPostConnector

Verbindet jedes Neuron der presynaptischen Population mit einer festen Anzahl Neuronen der postsynaptischen Population.

```
pynn.FixedNumberPostConnector(n = 10, allow_self_connections = True,
                              weights = 1.0)
```

n ist die Anzahl der Verbindungen pro Neuron der presynaptischen Population. Die anderen Parameter werden genauso verwendet wie bei *pynn.AllToAllConnector*.

2.3.3 pynn.FixedNumberPreConnector

Verbindet eine feste Anzahl Neuronen der presynaptischen Population mit jedem Neuron der postsynaptischen Population.

```
pynn.FixedNumberPostConnector(n = 10, allow_self_connections = True,
                              weights = 1.0)
```

n ist die Anzahl der Verbindungen pro postsynaptischem Neuron. Die anderen Parameter werden genauso verwendet wie bei *pynn.AllToAllConnector*.

2.3.4 pynn.FromListConnector

```
pynn.FromListConnector(conn_list = [(pre, post, weight, delay), (...), ...])
```

Mit der Liste *conn_list* können direkt Verbindungen zwischen dem *pre*-ten Neuron der presynaptischen Population und dem *post*-ten Neuron der postsynaptischen Population mit Gewicht *weight* und Delay *delay* erzeugt werden.
pre und *post* sind die Indizes der zu verbindenden Neuronen innerhalb ihrer Populationen (siehe auch *pynn.Projection*).
weight ist die jeweilige Stärke der Verbindung (siehe *Hardware Gewichte*)
delay ist der synaptische Delay in Millisekunden den Spikes haben sollen. Da Spikey nicht über konfigurierbare Delays verfügt ist hier immer *None* einzutragen.

2.3.5 pynn.OneToOneConnector

```
pynn.OneToOneConnector(weights = 1.0, delays = None)
```

Dieser Connector verbindet das 0-te Neuron der presynaptischen Population mit dem 0-ten Neuron der postsynaptischen Population, das 1-te mit dem 1-te und so weiter bis alle Neuronen der kleineren Population verbunden sind. Alle weiteren Neuronen bleiben ohne Verbindung.
Die Parameter sind analog zu den anderen Connectoren.

2.4 Sources

Sources sind externe Spike-Quellen, welche benutzt werden können um Neuronen zu stimulieren. Man kann entweder ein Array mit festen Spike-Zeitpunkten angeben (*pynn.SpikeSourceArray*) oder eine Quelle generieren, welche automatisch Spike-Zeitpunkte erzeugt die einer Poissonverteilung folgen (siehe auch *pynn.SpikeSourcePoisson*).

2.4.1 pynn.SpikeSourceArray

```
pynn.SpikeSourceArray(parameters = {'spike_times': [1, 10, 20]})
```

Erzeugt externe Spikes zu den Zeitpunkten 1ms, 10ms und 20ms. Die Spike-Zeitpunkte werden als Array oder Liste übergeben.

2.4.2 pynn.SpikeSourcePoisson

```
pynn.SpikeSourcePoisson(parameters = {'duration': 1000., 'rate': 1.0,
                                      'start': 0.0})
```

Erzeugt nach *start* ms (hier 0ms, also von Simulationsbeginn an) Spikes welche poissonverteilt sind mit einer Rate (Spike-Frequenz) von *rate* (hier 1.0Hz). Mit *duration* lässt sich einstellen, wie lange die Quelle Spikes generieren soll (hier für 1000ms).

2.5 Readout

Methoden welche aufgerufen werden müssen um Membranspannungen oder Spike-Zeitpunkte der Neuronen einer Population zu messen/aufzunehmen.

2.5.1 pynn.record

Wird aufgerufen um die Spikes einer Population aufzunehmen. Muss vor Beginn der Simulation (*pynn.run*) verwendet werden.

```
pop = pynn.Population(dims=5, cellclass = pynn.IF_facets_hardware1,
                      cellparams = params, label = '')
pop.record()
```

record ist eine Methode der Population-Klasse und muss nach Erstellen einer Population einfach nur aufgerufen werden.

2.5.2 pynn.getSpikes

Wird aufgerufen um die Spikes einer Population auszulesen.

```
pop = pynn.Population(dims=5, cellclass = pynn.IF_facets_hardware1,
                      cellparams = params, label = '')
pop.record()
pynn.run(1000)
spikes = pop.getSpikes()
```

spikes ist nun eine Liste bestehend aus Tupeln (Neuron Nr., Spike-Zeitpunkt), wobei Neuron Nr. angibt, welches Neuron der Population gemeint ist. Besteht die Population nur aus einem Neuron, ist der Neuron Nr. Eintrag identisch für alle Tupel. In diesem Fall kann man die Spike-Zeitpunkte wie folgt auslesen:

```
spikes = pop.getSpikes()[:,1]
```

Besteht eine Population aus mehreren Neuronen und man will nur die Spike-Zeitpunkte des z.B. dritten Neurons, geht dies wie folgt:

```
spikes = pop.getSpikes()
spikes = spikes[spikes[:,0]==2, 1]
```

2.5.3 pynn.record_v

```
pynn.record_v(pop[i], '')
```

Nimmt die Membranspannung des i-ten Neurons der *Population* pop auf (z.B. $i = 0$ wäre das erste Neuron der Population). **Achtung:** Auf Spikey kann in jedem *pynn.run* nur die Spannung von einem einzelnen Neuron aufgenommen werden.

2.5.4 pynn.timeMembraneOutput

```
times = pynn.timeMembraneOutput()
```

times beinhaltet die Zeitpunkte, zu denen die Membranspannung vermessen wurde.

2.5.5 pynn.membraneOutput

```
membrane = pynn.membraneOutput()
```

membrane beinhaltet die aufgenommenen Werte der Membranspannung, für die zugehörigen Zeiten siehe *pynn.timeMembraneOutput*.

2.6 Hardware Gewichte

Spikey implementiert 4-bit synaptische Gewichte. Eingestellte Gewichte die zwischen den realisierten liegen werden stochastisch gerundet. Die folgenden Parameter werden verwendet um die Stärke von synaptischen Gewichten als Vielfaches der maximal oder minimal einstellbaren Hardware-Gewichte festzulegen. Die einstellbaren Hardware-Gewichte unterscheiden sich für inhibitorische (Inh) und exzitatorische (Exc) Gewichte.

```
connector = pynn.AllToAllConnector(allow_self_connections = True,
                                   weights = 2 * pynn.minExcWeight)
pynn.Projection(presynaptic_population = pop1,
                postsynaptic_population = pop2,
                method = connector, target = 'excitatory')
```

In diesem Beispiel werden die Neuronen der Populationen pop1 und pop2 exzitatorisch verbunden, wobei das Gewicht dem doppelten minimalem Hardware-Gewicht entspricht (*2 * pynn.minExcWeight*).

2.6.1 pynn.maxExcWeight

Das maximal einstellbare exzitatorische Gewicht auf Spikey.

2.6.2 pynn.maxInhWeight

Das maximal einstellbare inhibitorische Gewicht auf Spikey.

2.6.3 pynn.minExcWeight

Das minimal einstellbare exzitatorische Gewicht auf Spikey.

2.6.4 pynn.minInhWeight

Das minimal einstellbare inhibitorische Gewicht auf Spikey.

2.7 Minimales Beispiel

```
# Erstelle pynn-Experiment
pynn.setup(mappingoffset=0)

# Erzeuge eine Poisson Spike-Quelle
# Erst legen wir die Parameter der Quelle fest
poisson_parameters = {'duration': 10000., 'rate': 500.0,'start': 0.0}
# Danach erzeugen wir die Spike-Quelle
pop1 = pynn.Population(dims=1, cellclass = pynn.SpikeSourcePoisson(poisson_parameters))

# Nun erstellen wir ein einzelnes Neuron
pop2 = pynn.Population(dims=1, cellclass = pynn.IF_facets_hardware1)
# Von diesem Neuron wollen wir den Membranverlauf aufnehmen
pynn.record_v(pop2[0])

# Zuletzt wird die Spike-Quelle mit dem Neuron verbunden
# Erst legen wir die Art der Verbindung fest
connector = pynn.AllToAllConnector(allow_self_connections = True,
                                   weights = 10 * pynn.minExcWeight)
# Danach verbinden wir die Quelle mit dem Neuron
pynn.Projection(presynaptic_population = pop1,
                postsynaptic_population = pop2,
                method = connector, target = 'excitatory')

# Nachdem das Experiment festgelegt ist (Quellen, Neuronen und Verbindungen),
# starten wir die Simulation
pynn.run(10000)

# Nach der Simulation lesen wir die aufgenommene Membranspannung aus
times = pynn.timeMembraneOutput()
membrane = pynn.membraneOutput()

# Beenden des Experiments
pynn.end()
```