Aufnahme von STDP[1]-Kurven mit Tripeln von Aktionspotentialen

Praktikumsbericht von Ole Schmidt

Durchgeführt vom 13.5. bis zum 28.6.
Betreuer: Thomas Pfeil
Arbeitsgruppe: Electronic Visions
Universität Heidelberg

[1]Spike-Time-Dependent Plasticity

Inhaltsverzeichnis

1 Einleitung

1.1 Theoretische Grundlagen

Morrison et al. (2008) geben eine Regel zur Gewichtsänderung $\Delta w^{\pm}$ von Synapsen, basierend auf der zeitlichen Abfolge von prä- und postsynaptischen Aktionspotentialen (im folgenden APs) an.

$$\Delta w^{+} = F_{+}(w) * exp(-\Delta t/\tau_{+}) \quad \text{für} \quad \Delta t > 0 \tag{1}$$
$$\Delta w^{-} = -F_{-}(w) * exp(\Delta t/\tau_{-}) \quad \text{für} \quad \Delta t \leq 0 \tag{2}$$

In Gleichung (1) beschreibt Δt den Abstand zwischen prä- und postsynaptischen APs $\Delta t = t_j - t_i$, wobei der Index i für das präsynaptische Neuron steht und j für das postsynaptische (siehe Abb. 1). $F_{\pm}(w)$ ist eine Funktion, die abhängig vom aktuellen Gewicht w die Gewichtsänderung skaliert. Die Zeitkonstanten $\tau_{\pm}$ beschreiben, wie lange sich die Synapse an das vorangegangene AP erinnert, also wie schnell die Spur eines APs verschwindet. Damit Gleichung (1) und (2) nach dem Protokoll in Abb. 1 unabhängig voneinander bleiben wird τ recht gewählt im Vergleich zum Abstand T zwischen den einzelnen AP-Paaren.

Es kommen unterschiedliche Gewichtsaktualisierungsregeln in Frage, wie z.B. die multiplikative Regel, die additive Regel oder Mischformen von beiden(*Morrison et al.*, 2008). In dieser Arbeit wurde nur die additive Gewichtsänderungsregel verwendet, somit ist $F_{\pm}$ unabhängig vom aktuellen Gewicht w und somit konstant. Dieses berechnet sich aus $F_{\pm} = w_{max} * A_{\pm}$. Da wir eine symmetrische STDP Kurve erhalten wollen, setzen wir $A = A_{+} = A_{-}$ und somit $F_{+} = F_{-}$. Außerdem werden die beiden Zeitkonstanten gleichgesetzt: $\tau = \tau_{-} = \tau_{+}$. Das Gewicht der Synapse wird also verstärkt, wenn das präsynaptische AP vor dem postsynaptischen AP die Synapse erreicht, also möglicherweise ein kausaler Zusammenhang zwischen beiden APs besteht. Ebenso wird das Gewicht für den umgekehrten Fall, dass ein postsynaptisches AP vor dem präsynaptischen AP eintrifft, verringert, da ein akausaler Zusammenhang besteht.

Die Variablen w_{min} und w_{max} legen den Bereich fest, in dem sich das Gewicht der Synapse bewegen kann. Sollte einer dieser Werte überschritten werden, wird das Gewicht wieder auf die entsprechende Grenze zurückgesetzt.

Eine STDP Kurve wird aufgenommen, indem eine minimale Gewichtsänderung w_{wc} festgelegt wird und wir messen, für wie viele Paare von APs N diese überschritten wird, d.h. es gilt dann:

$$w_{wc} \leqq N * w_{max} * A * exp(-|\Delta t|/\tau) \tag{3}$$

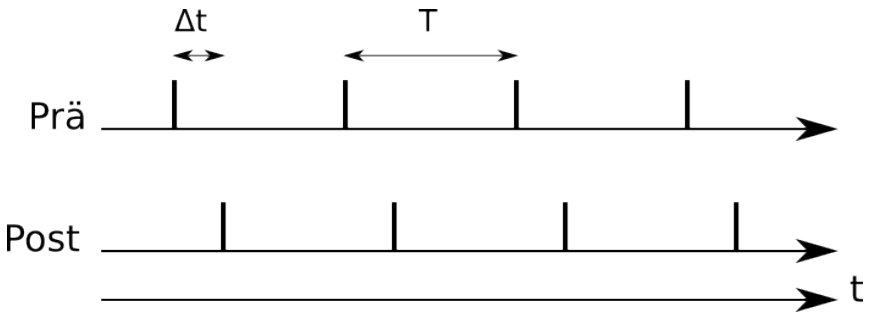

Abbildung 1: Zeitliche Abfolge von prä- und postsynaptischen APs. Als Beispiel werden $N = 4$ Paare dargestellt, periodisch im Abstand $T >> \Delta t$ um (1) und (2) unabhängig voneinander zu machen.

Die Gewichtsänderung ist also größer, je dichter die prä- und postsynaptischen APs beieinanderliegen, je mehr Paare von APs verwendet werden und hängt außerdem von einem konstanten Faktor $w_{max} * A$ ab, der sich über die Parameter für die Synapse einstellen lässt

Zur Aufnahme der STDP Kurven wird nun $\frac{1}{N}$ in Abhängigkeit von Δt gemessen und aufgetragen.

1.2 Ziel und Motivation

In der Bachelorarbeit *Scherzer* (2013) wurden unter anderem STDP Kurven auf dem Spikey Chip (*Pfeil et al.*, 2012) ausgemessen. Dabei entstanden große Plateaus im Bereich für kleine Abstände zwischen prä- und postsynaptischen APs, da ein AP-Paar immer ausreichend war, um die Schwelle der minimalen Gewichtsänderung zu übertreten (siehe Abb. 2). Dieses Praktikum beschäftig sich mit einer Methode zur Verbesesrung des Auflösevermögens, um die Plateaus zu vermeiden. Die Grundidee ist es, einen drittes präsynaptisches AP hinter dem postsynaptischen AP für kausale Gewichtsänderung, respektive vor dem postsynaptischen AP für akausale Gewichtsänderungen, zu setzen, um so die Anzahl der benötigten AP-Paare, bzw. Tripel zu erhöhen. Dies entspricht einer Erhöhung der Schwelle w_{wc}. Diese soll in der Hardware konstant gehalten werden, da dessen Verhalten noch nicht genau vermessen wurde.

So könnte ab dem Bereich, in dem das Plateau beginnt (Abb. 2: $|\Delta t| \lessapprox 5ms$), ein weiteres präsynaptisches AP hinzugenommen werden, um so die Schwelle künstlich zu erhöhen.

Um sich mit dem Computerprogramm, dass die STDP Kurve aufnimmt, auseinanderzusetzen, wurde für dieses zuerst die Hardwaremessung durch den Softwaresimulator NEST (*Gewaltig and Diesmann*, 2007) ersetzt und ein Umschalter eingebaut, um überprüfen zu können, ob die Grundidee richtig ist. So konnten nebenbei auch die Ergebnisse von NEST mit den theoretisch berechneten Werten verglichen und die Gewichtsregel überprüft werden.

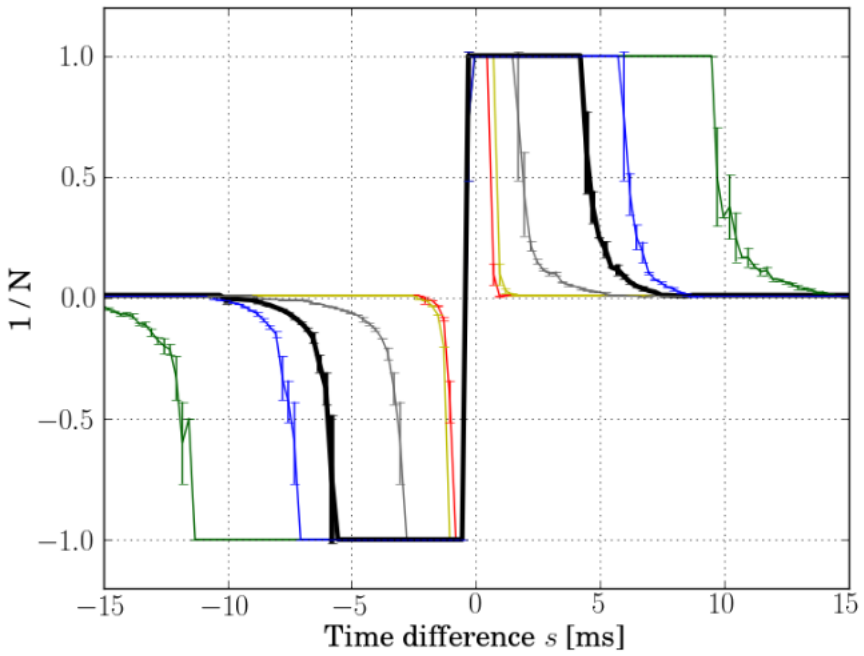

Abbildung 2: STDP Kurven auf Spikey aus*Scherzer* (2013). Aufgetragen sind 6 STDP-Kurven für unterschiedliche Parameter. Wir betrachten nur die dicke schwarze Kurve.

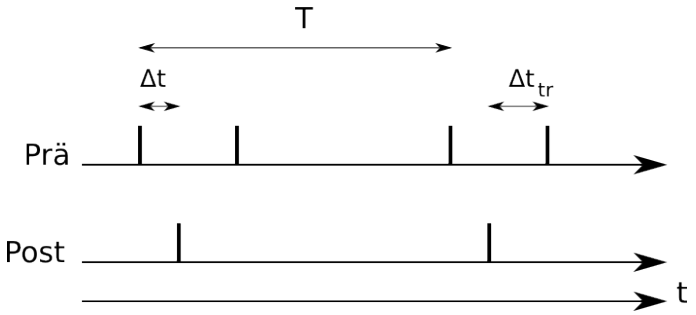

Abbildung 3: Zeitliche Abfolge von prä- und postsynaptischen APs mit Tripeln für $N = 2$.

2 Methoden

2.1 Grundidee: Hinzufügen eines zweiten präsynaptischen APs

Um das in der Motivation genannte Problem der zu niedrigen Schwelle w_{wc} zu lösen, wird nun ein drittes AP hinzugefügt. Die zeitliche Abfolge der APs sieht nun aus wie in Abb. 3 dargestellt. Der Abstand des zweiten präsynaptischen APs zum postsynaptischen AP muss immer größer sein, als der Abstand des ersten präsynaptischen APs zum postsynaptischen AP, damit sich das Gewicht nicht in die entgegengesetzte Richtung ändert: $\Delta t_{tr} > \Delta t \quad \forall \Delta t$.

Außerdem muss der Bereich eingestellt werden, ab dem das zusätzliche AP verwendet wird, da sonst für große Δt die Schwelle w_{wc} nur noch für extrem viele APs überschritten wird. Der Abstand des 2. präsynaptischen APs (Δt_{tr}) darf nicht zu hoch gewählt werden, da sonst der Einfluss bei kleinen Δt zu gering wird und man wieder ein Plateau erhält.

2.2 Aufbau der Software

Zur Aufnahme der STDP Kurven mit NEST wurde ein pyNN (*Davison et al.*, 2008) Skript verwendet, dass in mehrere Unterprogramme gegliedert ist. Das Hauptprogramm *(stdp_curve.py)* ruft das Messprogramm *(measure_corr.py)* auf und schreibt die Ergebnisse für Δt, N und deren Standardabweichungen in eine Datei zur späteren Analyse.

Ziel ist es, für alle Δt die Anzahl N an AP-Paaren zu finden, ab der sich das Gewicht ändert, für die also die Gleichung (3) erfüllt ist. Im Messprogramm werden für das gewünschte Δt und eine festgelegte Anzahl an AP-Paaren (bzw. Triplen) die Abfolge der APs generiert und an die Software-Implementierung *simulator.py* übergeben. Dieses ermittelt die Gewichtsänderung und gibt eine Variable zurück, je nachdem ob w_{wc} (siehe Gleichung (3)) überschritten wurde oder nicht. Falls das Gewicht sich nicht ausreichend geändert hat, wird die Messung mit einem anderen N wiederholt. Zur Ermittelung von N können zwei Methoden verwendet werden. Zum einen kann N iterativ erhöht werden, oder aber man verwendet eine Intervallmethode (siehe Abschnitt 2.4), um schneller auf größere N zu kommen und somit die Anzahl der Simulationen pro STDP Kurve drastisch zu verringern.

2.3 Software Simulationen

In der Software-Implementierung für den Simulator (Quellcode siehe Anhang) wird ein Integrate-and-Fire Neuron erstellt, dass an zwei Quellen von APs (sog. *SpikeSourceArrays*) angeschlossen ist. Die eine Quelle dient zur Stimulation des Neurons, um postsynaptische APs zu erzeugen, die andere sendet präsynaptische APs aus und ist über eine dynamische Synapse mit dem zu stimulierenden Neuron verbunden. Ist im folgenden von präsynaptischen APs die Rede, so sind

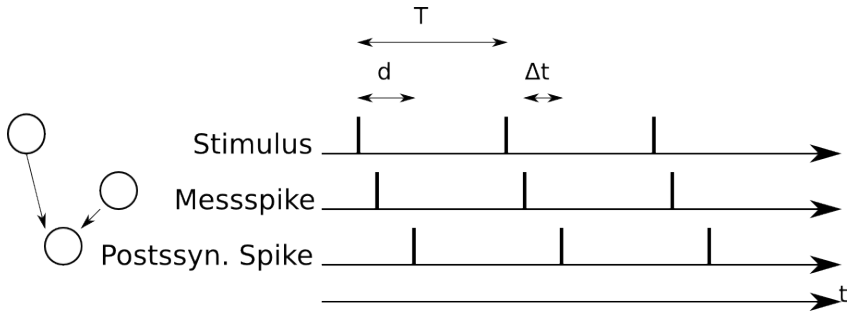

Abbildung 4: Zeitliche Abfolge von prä- und postsynaptischen APs im Simulator für $N = 3$.

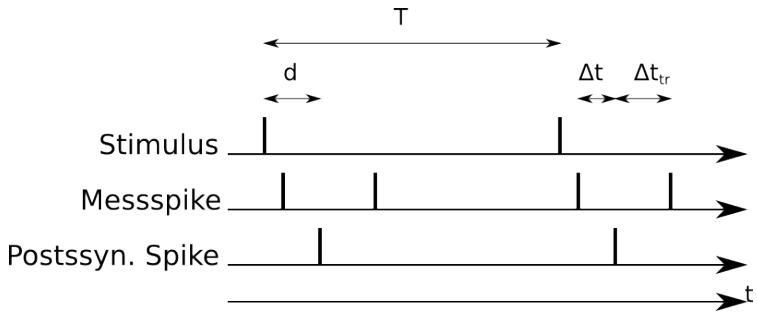

Abbildung 5: Zeitliche Abfolge von prä- und postsynaptischen APs mit Triplen für $N = 2$.

nur die APs aus der Quelle gemeint, die über die dynamische Synapse an das Neuron angeschlossen sind. Diejenigen, die es zum Feuern anregen sollen werden stimulierende APs genannt. Die stimulierende Quelle muss so 'stark' an das Neuron angeschlossen sein, dass ein einziges AP als Stimulation ausreicht, ein postsynaptisches AP auszulösen. Dahingegen darf das Startgewicht der dynamischen Synapse nicht zu groß gewählt werden, damit die präsynaptischen APs nicht ihrerseits das Neuron zusätzlich zum Feuern anregen.

Um dies zu überwachen, werden die Zeiten zu denen APs ausgesendet werden, zusätzlich ausgegeben, um die Verzögerung d zwischen Stimulus und postsynaptischem AP zu ermitteln. So kann Δt genau eingestellt werden. Die zeitliche Abfolge der einzelnen APs ist in Abb. 4 dargestellt.

Natürlich dürfen die unterschiedliche APs nicht zu dicht beieinanderliegen, da die Spur der vorigen APs sonst das Gewicht zusätzlich verändern würden. Als Abstand zwischen den AP-Paaren oder -Triplen wird jeweils $T = 100\,\text{ms}$ gewählt (Abb. 4 ist nur als Übersicht gemeint und nicht maßstabsgetreu), die Zeitkonstante τ wird zudem relativ klein gewählt, damit die Spur schnell verschwindet.

2.3.1 Simulation mit Paaren

Für einfache Paare ist das Timing der APs in Abb. 4 dargestellt. Im Vergleich zu Abb. 1 kommt der oben erwähnte Stimulus hinzu, der nach einer Verzögerung d ein postsynaptisches AP auslöst.

2.3.2 Ergänzung zu Tripeln

Um ein zweites präsynaptischen AP einzubauen, muss nur die Funktion *generateSpikeTrains* in *measure_corr.py* leicht abgeändert werden. So wird die zeitlichr Abfolge der APs des Stimulus unangetastet gelassen und zu jedem Paar von APs wird noch ein drittes präsynaptisches AP hinzugefügt. Die zeitliche Abfolge der APs sieht aus wie in Abb. 5 dargestellt.

Die Anzahl der benötigten AP-Tripel N_{tr} ergibt sich aus:

$$w_{wc} \leq N_{tr} * w_{max} * A * (exp(-|\Delta t|/\tau) - exp(-|\Delta t_{tr}|/\tau)) \tag{4}$$

2.3.3 Zurückrechnen auf Paare

Das Plateau kann nun feiner aufgelöst werden, jedoch ergeben die Messpunkte auch noch keine glatte STDP Kurve. Dafür müssen die Daten für den Bereich, in dem Tripel verwendet werden, ersteinmal zurückgerechnet werden auf das, was man nur mit AP-Paaren gemessen hätte. Daher wird N nicht gerundet, sodass Werte mit $N < 1$ möglich sind. Die Formel fürs zurückrechnen ergibt sich durch Gleichsetzen der Gleichungen (4) und (3). Das neue N berechnet sich dann zu:

$$N = N_{tr} * (1 - \frac{exp(-|\Delta t_{tr}|/\tau)}{exp(-|\Delta t|/\tau)}) \tag{5}$$

Diese Umrechnung ist im Analyseprogramm *(analysis.py)* implementiert, welches die Datei mit den Ergebnissen lädt und diese grafisch darstellt. So bleiben die eigentlichen Messdaten unangetastet.

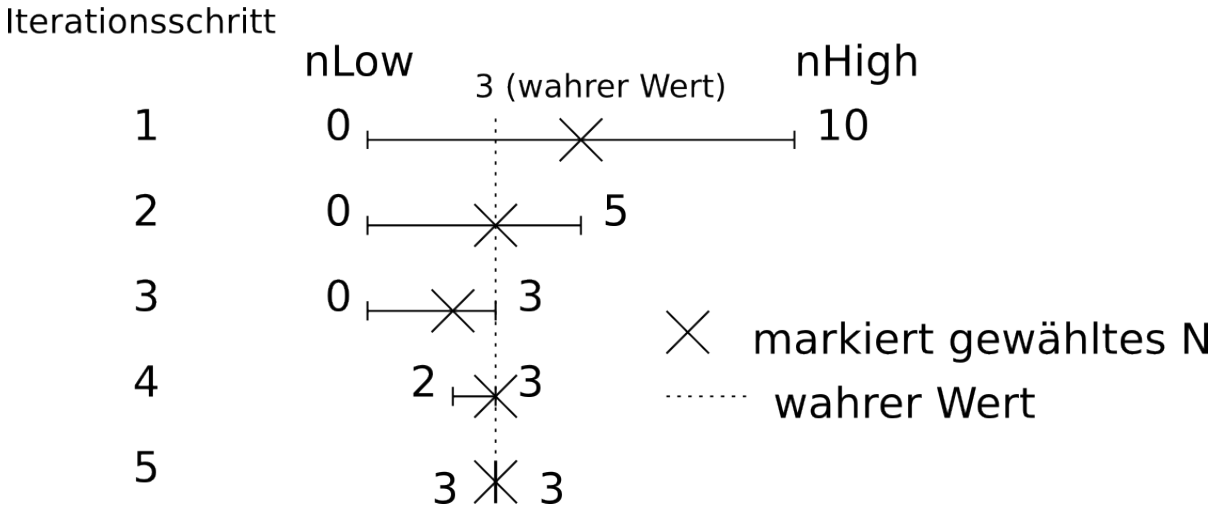

Abbildung 6: Funktionsweise der Intervall-Methode. Beschreibung siehe Abschnitt 2.4.

2.4 Intervallmethode

Um schnell auf die Anzahl der AP-Paare zu kommen, die für ein Gewichtsupdate ausreichen, wird eine einfache Intervallmethode (Algorithmus der Binären Suche) verwendet. Zuerst wird vorgegeben, in welchem Bereich N zu suchen ist:

$$nLow \leq N \leq nHigh \tag{6}$$

Falls noch kein Startwert vorgegeben ist, wird in der Mitte des Intervalls, also mit $N = \frac{nHigh}{2}$, begonnen. Reicht die Anzahl der AP-Paare nicht aus, um das Gewicht ausreichend zu verändern ist N zu klein und der vorige Wert wird als untere Grenze übernommen. Anschließend wird in dem neuen, halb so großen Intervall gesucht. Falls w_{wc} überschritten wurde, weiss man noch nicht, ob nicht auch eine kleineres N genügt hätte. Also wird der vorige Wert als Maximum übernommen und wiederum im neuen Intervall gesucht. So wird iterativ das Intervall halbiert bis der wahre Wert erreicht ist.

```
def intervalMethod(deltaT, nOld=None ):
  """Find number of spike pairs needed for a weight update by devide and conquer

  Keyword arguments:

  deltaT -- desired delay between pre- and postsynaptic spikes
  nOld -- start value for number of spike pairs"""

  nLow = 0 # Untere Grenze vom ersten Intervall
  nHigh = 130 # Obere Grenze vom ersten Intervall
  delay = measureConfig['delayBase'] # Zeit zwischen stimulierendem AP und postsynaptischen AP
  N = None
  flag = None # Variable, die anzeigt ob das Gewicht sich verändert hat

  # flag = 1 bedeutet postitives Gewichtsupdate, flag = 1 bedeutet negatives Gewichtsupdate
  # flag = 0 bedeutet gar kein Gewichtsupdate
  while True:
    #return condition
    if nHigh - nLow < 1: # nHigh soll gleich nLow sein!
      if flag == 2:      # Für negatives Gewichtsupdate muss N mit -1 multipliziert werden
        N = -N
      return n, deltaTMean, deltaTStd, overallTrials, failedTrials

    #devide
    if N != None or nOld == None:
      N = nLow + round((nHigh - nLow) / 2.0)
    else:
      N = nOld

    flag, delay = run(N, deltaT, delay)

    #conquer
    if flag == 0:
      nLow = n
    elif flag != 0 and nHigh - nLow == 1:  #Beträgt der Abstand nur noch 1, muss nLow erhöht werden,
      nLow += 1                            #sonst ist die Abbruchbedingung nie erfüllt.
    else:
      nHigh = n
```

Abbildung 7: Quellcode Intervall-Methode.

Wenn minimaler und maximaler Wert für N nahe beieinander liegen, also $nHigh - nLow < 1$, ist das richtige N gefunden. Dies würde allerdings so nicht funktionieren, da der Abstand zwischen $nHigh$ und $nLow$ nie kleiner als 1 werden kann. Daher muss die untere Grenze um 1 erhöht werden, wenn $nHigh - nLow = 1$ gilt.

In Abb.6 ist dargestellt, wie der wahre Wert $N = 3$ mit der Intervall-Methode im Bereich von 0-10 gefunden werden würde. Das Intervall ist immer durch den Balken dargestellt und das als nächstes verwendete N durch das Kreuz auf dem Balken markiert. Dass das gewählte N nicht immer in der Mitte des Intervalls liegt, liegt daran, dass $\frac{nHigh}{2}$ gerundet wird.

3 Ergebnisse

3.1 STDP Kurven mit Paaren von APs

Zuerst einmal wurden STDP Kurven für Paare von APs aufgenommen, und dabei mit unterschiedlichen Parametern experimentiert. Um den Vergleich mit der Arbeit (*Scherzer*, 2013) zu ermöglichen, wurde $\tau = 5\,\text{ms}$ gewählt. A und w_{max} wurden konstant gehalten und nur w_{wc} verändert, da diese drei Faktoren wie in Gleichung (4) zu sehen linear zusammenhängen. Es ist deutlich zu sehen, dass für eine kleine Schwelle ($w_{wc} = 0.0001$) ein Plateau entsteht. Alle Parameter sind direkt in die Plots eingetragen.

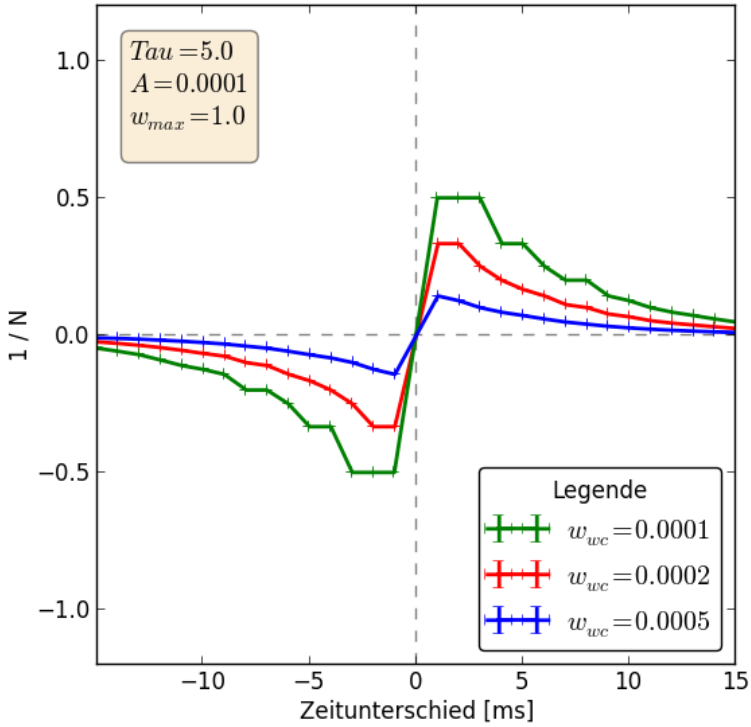

Abbildung 8: STDP Kurven für Paare von APs, gemessen mit unterschiedlich hoher Schwelle w_{wc}. Die unterschiedlichen Farben entsprechen unterschiedlich hohen Schwellen.

3.2 Auflösen des Plateaus

Wie in der Abb. 8 zu sehen ist, entsteht für eine kleine Schwelle ein Plateau in der Region um $|\Delta t| \lessapprox 3\,\text{ms}$. Nimmt man zu jedem AP-Paar ein drittes AP hinzu (siehe schematische Abb. 5), lässt sich das Plateau auflösen. In Abb. 9 wird dies dargestellt. Im Bereich von $-6\,\text{ms} < \Delta t < 6\,\text{ms}$ wurden das dritte AP hinzugefügt im Abstand zum postsynaptischen AP von $\Delta t_{tr} = 6.5\,\text{ms}$

Wird mit diesen Einstellungen eine STDP Kurve aufgenommen, so macht die Kurve bei den Übergängen von Tripeln von APs zu Paaren Sprünge. In diesem Bereich muss das N auf Paare von APs zurückgerechnet werden, um eine durchgehende Kurve zu erhalten.

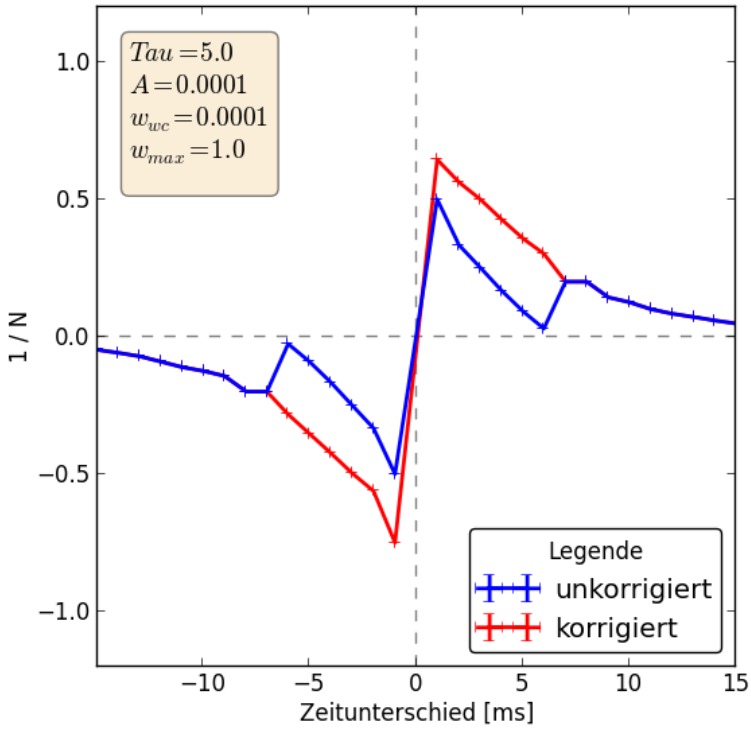

Abbildung 9: STDP Kurve aufgenommen mit Tripeln von APs, einmal unkorrigiert, einmal zurückgerechnet auf Paare.

3.3 Zurückrechnen der Daten auf auf AP-Paare

Wie in der Abb. 9 zu sehen, kommen für N jetzt nicht mehr nur ganze Zahlen in Frage. Für $\Delta t = 1\,\text{ms}$ liegt N zwischen 1 und 2.

3.4 Vergleich mit theoretischen Kurven

Lässt man sich die Gleichungen (3) und (4) direkt mit Python ausrechnen, ohne einen Simulator zu benutzen, kann man die Ergebnisse der Messung mit dem Software Simulator mit dem vergleichen, was man in der Theorie erwartet. In Abb. 10 ist dieser Vergleich dargestellt. Für die rote, ganz glatte, Kurve in der Abb. 10 wird N theoretisch bestimmt und nicht gerundet. Das Plateau, welches bei einfachen AP-Paaren entsteht wird durch die Ergänzung der dritten APs vermieden und die Kurve gleicht sich mehr der Theorie an. Dieser Graph verifiziert die erprobte Methode und zeigt die Verbesserung im Vergleich zu der Verwendung von AP-Paaren.

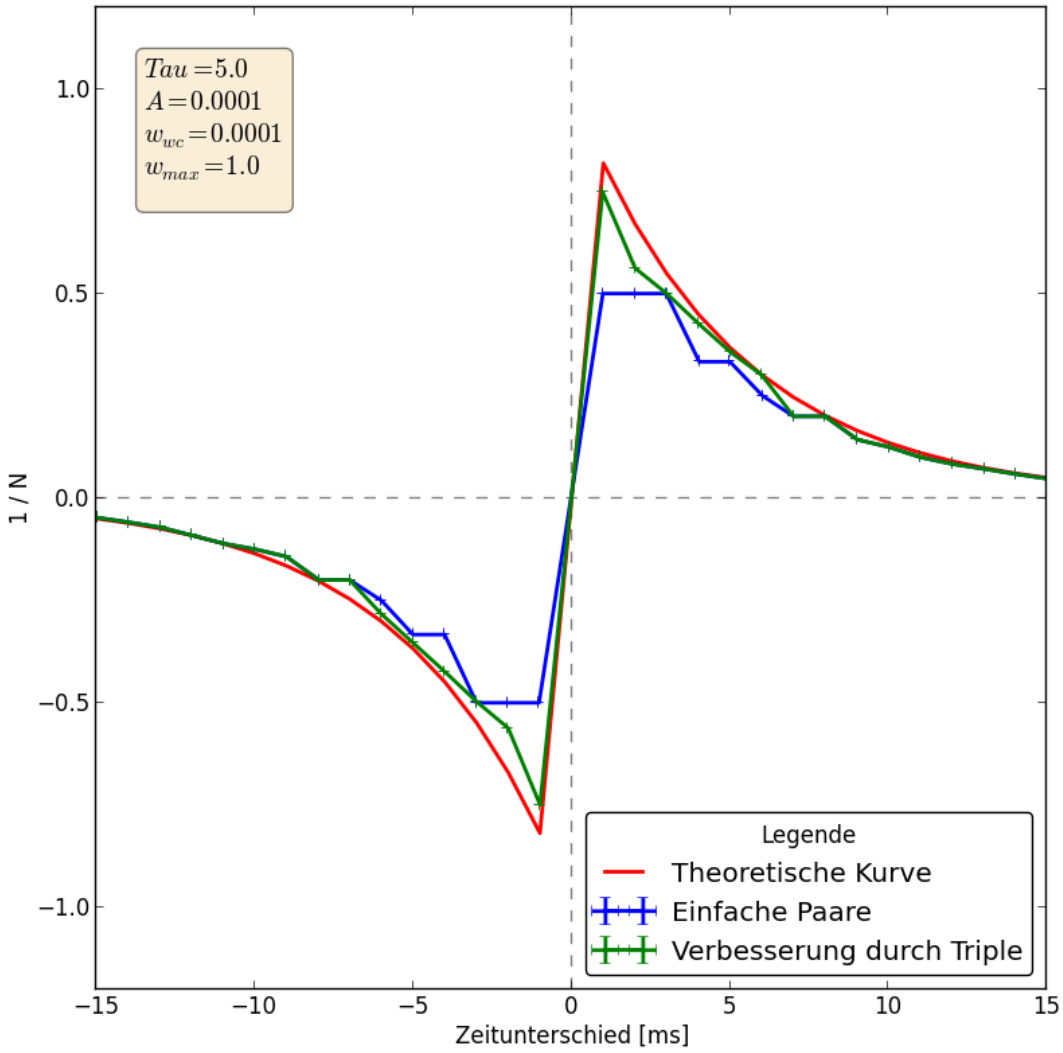

Abbildung 10: Vergleich der verschiedenen STDP Kurven. Gemessen mit Paaren von APs entsteht das Plateau. Mit Tripeln von APs gleicht sich die STDP Kurve mehr der theoretischen Referenz an.

3.5 Unit-Tests

Um die Ergebnisse besser vergleichen zu können, wurden *Unit Tests* mit Python erstellt, die die theoretisch ermittelten Werte mit den experimentell mit NEST bestimmten vergleicht. Zusätzlich wird die aus der Intervallmethode resultierende STDP Kurve mit der aus der iterativen Methode resultierenden verglichen.

In den Test sind *assertions* eingebaut, die einen Fehler hervorrufen wenn die auf unterschiedliche Methoden bestimmten N um mehr als 1 voneinander abweichen. Nur wenn N sehr groß wird, z.B. weil Δt sehr groß ist oder Tripel verwendet werden, wird eine größere Abweichung von max. 10% toleriert. Diese Tests laufen ohne Probleme durch, womit zum einen beide Methoden validiert sind und zum anderen auch Experiment mit Theorie ausreichend übereinstimmt. Außerdem kann so in Zukunft, wenn zum Beispiel auf Hardware gemessen wird, ebenfalls schnell

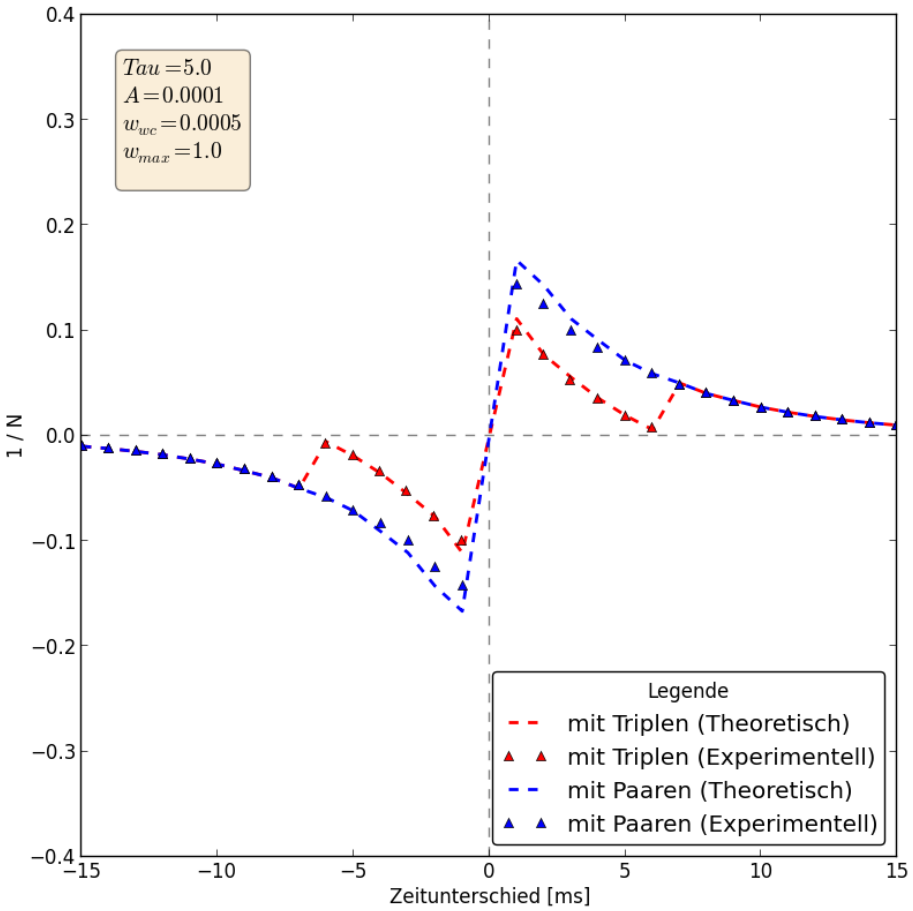

Abbildung 11: Graphische Darstellung der Differenz zwischen Theorie und Messung. Die Tripel wurden nicht korrigiert, um beide Kurven voneinander trennen zu können.

überprüft werden, ob die Abweichungen für das bestimmte N von der theoretischen Vorhersage zu groß werden.

Als zusätzlicher graphischer Vergleich sind nochmals die experimentellen Werte zusammen mit den theoretischen in einem Plot (siehe Abb. 11) dargestellt.

4 Zusammenfassung

Alles in allem kann man sagen, dass die Idee der Tripels von APs gut funktioniert. Da die Funktion erfolgreich in NEST implementiert werden konnte, spricht nichts dagegen, dass dies auch auf der Hardware funktioniert. Dies soll unter anderem im Rahmen einer Bachelorarbeit auf dem Spikey Chip vermessen werden.

Desweiteren konnte eine Intervallmethode zur Auffindung vom richtigen N verifiziert werden, mit deren Hilfe sich N deutlich schneller finden lässt als mit der einfachen iterativen Methode. Dies hat mit dem Software-Simulator noch keine große Rolle gespielt, da die Laufzeit der Messung mit Software sehr viel geringer ist als mit Hardware. So kann dieses auf der Hardware sehr hilfreich sein.

Durch die *Unit Tests* kann zudem überprüft werden, ob experimentelle Werte zu stark von der Theorie abweichen.

5 Anhang

```
  # stimulated neuron
  neuron = pynn.Population(1, pynn.IF_cond_exp)

  lastInputSpike = np.max(np.concatenate((stimulus, stimulusMeasure)))
  runtime = lastInputSpike
  stdpFrequency = runtime

  spikeSourceMeasure = pynn.Population(1, pynn.SpikeSourceArray, {'spike_times': stimulusMeasure})
  spikeSourceStim = pynn.Population(1, pynn.SpikeSourceArray, {'spike_times': stimulus})

  # connect and record
  stdp_model = pynn.STDPMechanism(timing_dependence=pynn.SpikePairRule(tau_minus=ParamConfig['tau_minus'],
['tau_plus']), weight_dependence=pynn.AdditiveWeightDependence(w_min=ParamConfig['w_min'], w_max=ParamConf
A_plus=ParamConfig ['A_plus'], A_minus=ParamConfig['A_minus']))

  pynn.Projection(spikeSourceStim, neuron, pynn.AllToAllConnector(weights=0.1), target='excitatory')
  prj = pynn.Projection(spikeSourceMeasure, neuron,
                        method=pynn.AllToAllConnector(weights=ParamConfig['startweight']),
                        target='excitatory',
                        synapse_dynamics=pynn.SynapseDynamics(slow=stdp_model))
  neuron.record()
```

Abbildung 12: Quellcode vvom Simulator.

Literatur

Davison, A. P., D. Brüderle, J. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger, PyNN: a common interface for neuronal network simulators, *Front. Neuroinform.*, *2*(11), 2008.

Gewaltig, M.-O., and M. Diesmann, NEST (NEural Simulation Tool), *Scholarpedia*, *2*(4), 1430, 2007.

Morrison, A., M. Diesmann, and W. Gerstner, Phenomenological models of synaptic plasticity based on spike timing, *Biological Cybernetics*, *98*(6), 459–478, doi:10.1007/s00422-008-0233-1, 2008.

Pfeil, T., T. C. Potjans, S. Schrader, W. Potjans, J. Schemmel, M. Diesmann, and K. Meier, Is a 4-bit synaptic weight resolution enough? – Constraints on enabling spike-timing dependent plasticity in neuromorphic hardware, *ArXiv e-prints*, 2012.

Scherzer, A.-C., Phase-locking on neuromorphic hardware, Bachelor thesis (German), University of Heidelberg, HD-KIP 10-43, 2013.