DISSERTEAÇÃO DE MESTRADO Nº 433

UMA REVISÃO DE MODELOS
E APLICAÇÕES DE
NEURÔNIOS PULSANTES

Roberto Coelho de Berredo

DATA DA DEFESA: 25/11/2005
Universidade Federal de Minas Gerais

Escola de Engenharia

Programa de Pós-Graduação em Engenharia Elétrica

UMA REVISÃO DE MODELOS E APLICAÇÕES
DE NEURÔNIOS PULSANTES

Roberto Coelho de Berredo

Dissertação de Mestrado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito para obtenção do Título de Mestre em Engenharia Elétrica

U. F. M. G. - BIBLIOTECA UNIVERSITÁRIA

213130607

Não danifique esta etiqueta

Orientador: Prof. Antônio de Pádua Braga

61 54 98

Belo Horizonte - MG

Novembro de 2005
"UMA REVISÃO DE MODELOS E APLICAÇÕES DE NEURÔNIOS PULSANTES"

ROBERTO COELHO DE BERREDO

Dissertação de Mestrado submetida à banca examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais, como parte dos requisitos necessários à obtenção do grau de Mestre em Engenharia Elétrica.

Aprovada em 25 de novembro de 2005.

Por:

Antônio de Pádua Braga - Dr.
Prof. DELT/EEUFMG - orientador

Carmen Déa Moraes Pataro - Dra.
Profa. DELT/EEUFMG

Petr I. Ekel - Dr.
Prof. PUC/MG

Frédéric Alexandre - Dr.
Prof. INRIA/FRANÇA
Abstract

Although the nervous system and nervous cells have been studied since ancient times, almost all the important advances on the knowledge base took place along the past century, when neuroscience evolved and developed more complex and detailed neuron models [yC03]. However, engineering applications were - and still are - practically limited to the basic neuron models, like the MCP [MP43] and its variations. In traditional artificial neural networks, the neuron behavior is described only in terms of firing rate, while most real neurons, commonly known as spiking neurons, transmit information by means of pulses, also called action potentials or spikes. From these considerations a major question raises immediately: if we were able to build powerful applications in all fields of engineering using these simple models, what would it be possible to do with the more complex models? This question is the fundamental motivation of the present work.

Given the importance of more realistic neuron models, our main objective is to present a general and comprehensive overview of spiking neurons, ranging from biological neuron features to examples of practical applications. It will be also demonstrated how analog information can be coded by precisely timed spikes, emitted by different neurons, and how this coded information can be processed to produce usefull results [Maa97]. The aim of the present work is therefore to highlight what we believe will be one of the main components of the future computing machines: the spiking neuron.

As a contribution, besides the review itself, we present also a novel approach to the spiking neuron network architecture used in practical applications [Ruf98, BPK02], replacing the multiple synapse construction with single synapses, and thus enhancing the computational efficiency of the network.
Resumo

Embora os sistemas e as células nervosas tenham sido estudados desde os mais remotos tempos, todos os importantes avanços na base de conhecimentos tiveram lugar ao longo do século passado, quando a neurociência evoluiu e desenvolveu modelos de neurônios mais complexos e detalhados [YC03]. Entretanto, as aplicações de engenharia estiveram — e ainda estão — praticamente limitadas aos modelos mais básicos de neurônios, como o MCP [MP43] e suas variações. Nas redes neurais artificiais tradicionais o comportamento dos neurônios é descrito apenas em termos de taxa de disparo, enquanto a maioria dos neurônios reais, comumente denominados neurônios pulsantes, transmitem a informação por meio de pulsos, também chamados de potencial de ação. Destas considerações surge imediatamente uma importante questão: se somos capazes de construir poderosas aplicações em todos os campos da engenharia usando estes modelos simples, o que não seria possível de se realizar com os modelos mais complexos? Esta questão é a motivação fundamental do presente trabalho.

Dada a importância dos modelos mais realísticos, nosso principal objetivo é apresentar um panorama geral e abrangente de neurônios pulsantes, desde as características do neurônio biológico até exemplos de aplicações práticas. Será também demonstrado como informações analógicas podem ser codificadas por pulsos precisamente espaçados no tempo, emitidos por diferentes neurônios e como esta informação codificada pode ser processada para produzir resultados úteis [Maa97]. A meta do presente trabalho é, portanto, destacar o que acreditamos será um dos principais componentes das futuras máquinas computacionais: o neurônio pulsante.

Como contribuição, além da própria revisão, apresentamos também uma nova abordagem à arquitetura das redes de neurônios pulsantes usada em aplicações práticas [Ruf98, BPK02], substituindo a construção com sinapses múltiplas por sinapses singelas e, assim, melhorando a eficiência computacional da rede.
<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sumário</td>
<td>iv</td>
</tr>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>Sumário</td>
<td>iv</td>
</tr>
<tr>
<td>Lista de Figuras</td>
<td>v</td>
</tr>
<tr>
<td>Dedicatória</td>
<td>vi</td>
</tr>
<tr>
<td>1 Introdução</td>
<td>1</td>
</tr>
<tr>
<td>2 Neurônios Biológicos e Características Neuronais</td>
<td>3</td>
</tr>
<tr>
<td>3 Modelos de Neurônios Pulsantes</td>
<td>10</td>
</tr>
<tr>
<td>4 Neurodinâmica</td>
<td>16</td>
</tr>
<tr>
<td>5 Uma Aplicação Prática Usando RNP</td>
<td>18</td>
</tr>
<tr>
<td>6 Conclusões</td>
<td>25</td>
</tr>
<tr>
<td>Referências Bibliográficas</td>
<td>26</td>
</tr>
</tbody>
</table>
Lista de Figuras

1.1 Separação de dez grupos ........................................... 2
2.1 Diagrama genérico ................................................... 3
2.2 Sinapse ................................................................. 4
2.3 PPSE e PPSI ........................................................... 5
2.4 Exemplo de um PPSE ................................................ 6
2.5 Pulso do MRP .......................................................... 7
2.6 Características neuro-computationais ............................. 8
3.1 Neurônio integra-e-dispara ......................................... 10
3.2 Modelo de Hodgkin-Huxley ....................................... 12
3.3 Parâmetro do modelo HH .......................................... 14
3.4 Circuito equivalente de um modelo compartimental ........ 15
4.1 Histerese ................................................................. 17
5.1 Codificação por campos receptivos ............................... 19
5.2 Função de aprendizado ............................................. 20
5.3 Sinapse Multipla ...................................................... 21
5.4 Resultados ............................................................. 22
5.5 Clusterização de imagem ........................................... 23
5.6 PPSE equivalente ..................................................... 24
À minha família, professores e amigos, pelo apoio e paciência.
Capítulo 1

Introdução

Embora os sistemas e as células nervosas tenham sido estudados desde os mais remotos tempos, praticamente todos os importantes avanços nesta área de conhecimento tiveram lugar ao longo do século passado [yC03] e foram fundamentados em cinco disciplinas experimentais: anatomia, embriologia, fisiologia, farmacologia e psicologia [KS00]. Então, a partir de meados do século passado, a engenharia foi adotada como a sexta disciplina e, reciprocamente, a neurociência foi também adotada pela engenharia, ensejando todo o desenvolvimento da inteligência computacional e fazendo deste um tema essencialmente multidisciplinar.

Entretanto, enquanto a neurociência evoluiu e produziu modelos de neurônios detalhados e complexos, as aplicações de engenharia estiveram – e ainda estão – virtualmente limitadas aos modelos mais básicos de neurônios, como o MCP [MP43] e suas variações. Destas considerações origina-se imediatamente uma importante questão: se somos capazes de construir aplicações poderosas em todos os campos da engenharia usando estes modelos simples, o que não seria possível realizar com os modelos mais complexos?

Do ponto de vista da engenharia, fica claro que a resposta para esta e outras questões somente será possível com uma compreensão mais profunda dos neurônios biológicos e de como estes podem realizar computações rápidas e confiáveis [MB98]. Esta aparente dificuldade deve ser antes encarada como uma oportunidade de usar a temporização dos pulsos como uma variável adicional no processamento de informações em redes neurais [Boh04]. Existem todavia alguns problemas ainda não completamente resolvidos, apesar de todo o progresso alcançado nos últimos duas décadas, p.ex., como neurônios reais transmitem informações ou como usar a temporização dos pulsos para processar informações eficientemente [Nat96, GK02b].

Nos últimos dez anos nota-se uma mudança na comunidade de redes neurais artificiais, no sentido de enfatizar mais as redes neurais com neurônios pulsantes – RNP [TH86, Hop88, Hop95, Nat96]. Esta mudança, além do grande aumento na capacidade computacional, foi devida à possi-
bilidade do uso da temporização dos pulsos como parte essencial da computação em redes neurais [Maa97, Ger01, SARC03], artificiais ou não.

Dada a importância dos modelos mais realísticos de neurônios, o principal objetivo deste trabalho é apresentar uma visão abrangente dos neurônios pulsantes, desde as características dos neurônios biológicos até a apresentação de exemplos de aplicações práticas. Com isso pretende-se ressaltar o que entendemos será um dos componentes principais dos futuros computadores: o neurônio pulsante.

Figura 1.1: A RNP à esquerda foi capaz de separar corretamente dez grupos, mostrados à direita.

No Capítulo Dois são apresentadas as características gerais dos neurônios reais [yC03] e outros componentes dos sistemas nervosos, bem como suas propriedades básicas. No Capítulo Três são discutidos os modelos matemáticos de aplicação mais usual, como o modelo de Hodgikin-Huxley [HH52]. No Capítulo Quatro são implementados alguns dos modelos apresentados no capítulo anterior, com simulações do comportamento dinâmico dos neurônios reais.

No Capítulo Cinco é apresentada uma aplicação prática utilizando neurônios pulsantes e codificação temporal de informação, com a construção de uma RNP para executar tarefas de clusterização, como mostrado na figura 1.1. Como contribuição deste trabalho, é proposta uma arquitetura alternativa para realizar a mesma tarefa, mas com um menor custo computacional.
Capítulo 2

Neurônios Biológicos e Características Neuronais

Os neurônios são a unidade elementar de processamento e são conectados entre si de forma intricada e existem em diversos tamanhos e formas. Um neurônio genérico é composto por quatro partes funcionais: a árvore dendrítica, o soma, o axônio e as sinapses, como mostrado na figura 2.1 [yc03]. De uma forma geral, os sinais são coletados pela árvore dendrítica e levados ao soma. Se a entrada total ultrapassa determinado limite, um pulso é emitido e transmitido pelo axônio até outros neurônios. A conexão entre o axônio e os dendritos é chamada de sinapse e é onde ocorre o reforço ou atenuação do sinal recebido, através de um processamento não-linear [KS00].

Figura 2.1: Diagrama genérico simplificado de um neurônio (baseado em [yc03]).

Os sinais neuronais se apresentam na forma de sequências de pulsos, chamadas de trem de pulsos. A duração de um pulso é tipicamente da ordem de 1 – 2ms com uma amplitude de 100mV [GK02b, PAF+04], como mostrado na figura 2.3. Como todos os pulsos são semelhantes, sua
forma não contém nenhuma informação. É a quantidade e a temporização dos pulsos que contém informações [MB98].

O potencial elétrico através da membrana das células nervosas é gerado por diferentes concentrações de íons, dentro e fora das células. Cada membrana tem seu potencial de repouso, usualmente entre $-40$ e $-90mV$ [PAF$^+$04]. A concentração dos íons é regulada por transportadores ativos (bombas de íons) e canais de íons.

![Diagrama de uma sinapse](image)

Figura 2.2: Diagrama de uma sinapse, com suas vesículas, neurotransmissores e neuro-receptores.

A sinapse mais comum nos cérebros de vertebrados é a sinapse química [PAF$^+$04, KS]$^00$, que atua por processos bioquímicos, neurotransmissores e receptores para alterar o potencial de membrana. Como mostrado na figura 2.2, as sinapses químicas consistem de [PAF$^+$04]:

- terminação pré-sínaptica,
- terminação pós-sínaptica,
- fenda sinaptica (20 nm).

O potencial de membrana é diretamente afetado pelos potenciais pós-sínapticos – PPS gerados pelos pulsos recebidos dos neurônios pré-sínapticos. Se o potencial de membrana alcança um determinado limiar, um pulso é disparado e transmitido pelo axônio. Um PPS positivo é chamado de excitatório (PPSE) e um negativo, de inibitório (PPSI), como na figura 2.3.

Um neurônio pré-sínaptico é representado por $i$ e um pós-sínaptico, por $j$. O conjunto de todos os neurônios pré-sínapticos ao neurônio $j$ é definido como $\Gamma_j = \{i|i$ presynaptic to $j\}$. Da
Figura 2.3: Exemplos de pulso (esquerda) e PPSE e PPSI (direita).

mesma forma, o conjunto de todos os neurônios pós-sinápticos ao neurônio \( i \) é definido como \( \Gamma_i = \{ j | j \) postsynaptic to \( i \} \).

O instante em que um dado neurônio \( j \) emite um pulso será referido como o instante de disparo deste neurônio e será representado por \( t_i^{(f)} \), com \( f \) variando de 1 a \( n \). O trem de pulsos do neurônio \( j \) é então caracterizado pelo conjunto de instantes de disparo, dado pela equação.

\[
E_j = \{ t_j^{(1)}, t_j^{(2)}, ..., t_j^{(n)} \}
\] (2.1)

onde \( t_j^{(n)} \) é o mais recente pulso do neurônio \( j \), também representado por \( i \).

É importante considerar que, devido a distâncias interneurais e tempos de transmissão axonal finitos, haverá um atraso entre a emissão de um pulso e o início do correspondente PPS. A equação 2.2 mostra uma formulação matemática para um PPS genérico originado no neurônio \( j \) por um pulso emitido pelo neurônio \( i \) [GK02b].

\[
e_{ij}(t) = \left[ \exp \left( -\frac{t - t_i^{(f)} - \Delta_{ij}^{ax}}{\tau_m} \right) - \exp \left( -\frac{t - t_i^{(f)} - \Delta_{ij}^{ax}}{\tau_s} \right) \right] \mathcal{H}(t - t_i^{(f)} - \Delta_{ij}^{ax})
\] (2.2)

ou em uma forma mais simples [Boh03]

\[
e_{ij}(t) = \frac{t - t_i^{(f)} - \Delta_{ij}^{ax}}{\tau} \exp \left( -\frac{t - t_i^{(f)} - \Delta_{ij}^{ax}}{\tau} \right) \mathcal{H}(t - t_i^{(f)} - \Delta_{ij}^{ax})
\] (2.3)

onde \( \tau_m, \tau_s \) e \( \tau \) são constantes de tempo, \( \Delta_{ij}^{ax} \) é o atraso de transmissão axonal e \( \mathcal{H}(\cdot) \) é a função de Heaviside. A figura 2.4 é um exemplo da equação (2.3).
Figura 2.4: Exemplo de um PPSE usando a equação 2.3, com \( t_i^{(f)} = 4\, ms \), \( \Delta_{ij}^{\text{ax}} = 6\, ms \) e \( \tau = 4\, ms \).

Imediatamente após a emissão de um pulso o potencial de membrana não retorna diretamente ao potencial de repouso \( u_{\text{rest}} \), mas passa por uma fase de hiperpolarização abaixo do valor de repouso [GK02b], como mostrado na figura 2.5.

No equilíbrio a voltagem gerada pelas diferentes densidades dos íons é dada pela equação (2.4), conhecida como Equação de Nernst [Wil99].

\[
E = \frac{kT}{q} \ln \left( \frac{C_{\text{in}}}{C_{\text{out}}} \right)
\]  

(2.4)

onde \( E \) é o potencial de equilíbrio (também chamado de potencial de Nernst), \( k \) é a constante de Boltzmann, \( T \) é a temperatura em graus Kelvin, \( q \) é a carga do íon e \( C_{\text{in}} \) e \( C_{\text{out}} \) são as respectivas concentrações iônicas, dentro e fora da membrana.

Como vários íons estão simultaneamente presentes o potencial de repouso \( u_{\text{rest}} \) é determinado pelo equilíbrio dinâmico entre o fluxo de íons através dos canais e pelo transporte ativo de íons pelas bombas de íons. No caso de mais de um íon, a equação (2.4) é substituída pela equação (2.5), que leva em conta a permeabilidade da membrana para cada íon envolvido [PAF04, KS00].

\[
E = 58 \log \frac{P_X[K]_2 + P_{Na}[Na]_2 + P_{Cl}[Cl]_1}{P_X[K]_1 + P_{Na}[Na]_1 + P_{Cl}[Cl]_2}
\]  

(2.5)

\( P \) é a permeabilidade da membrana e os índices 1 e 2 indicam a concentração iônica dentro e
Figura 2.5: a- o potencial de membrana aumenta do potencial de repouso \( u_{\text{rest}} \) e atinge o limiar \( \theta \), b- período refratário absoluto: um pulso é gerado, tornando impossível o disparo de outro pulso, c- período refratário relativo: a hiperpolarização decresce e o potencial de membrana atinge novamente o potencial de repouso.

fora da membrana, respectivamente. A tabela 2 mostra a concentração intracelular dos principais íons.

A figura 2.6 ilustra as vinte principais características dos neurônios reais, considerando a injeção de pulsos simples [Izh04].
Figura 2.6: Exemplos de características neuro-computacionais. A corrente de entrada é plotada abaixo da resposta neural e a barra horizontal representa 20ms. Todas as características aqui apresentadas foram obtidas com um modelo simplificado de neurônio pulsante [Izh03]. Uma versão eletrônica desta figura e permissões de reprodução encontram-se em www.izhikevich.com.
<table>
<thead>
<tr>
<th></th>
<th>Concentração (mM)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Intracelular</td>
<td>Extracelular</td>
</tr>
<tr>
<td><strong>Lula</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Potássio ($K^+$)</td>
<td>400</td>
<td>20</td>
</tr>
<tr>
<td>Sódio ($Na^+$)</td>
<td>50</td>
<td>440</td>
</tr>
<tr>
<td>Cloro ($Cl^-$)</td>
<td>40-150</td>
<td>560</td>
</tr>
<tr>
<td>Calcio ($Ca^{2+}$)</td>
<td>0.0001</td>
<td>10</td>
</tr>
<tr>
<td><strong>Mamíferos</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Potássio ($K^+$)</td>
<td>140</td>
<td>5</td>
</tr>
<tr>
<td>Sódio ($Na^+$)</td>
<td>5-15</td>
<td>145</td>
</tr>
<tr>
<td>Cloro ($Cl^-$)</td>
<td>4-30</td>
<td>110</td>
</tr>
<tr>
<td>Calcio ($Ca^{2+}$)</td>
<td>0.0001</td>
<td>1-2</td>
</tr>
</tbody>
</table>

Tabela 2.1: Concentrações intra- e extracelulares para os principais íons.
Capítulo 3

Modelos de Neurônios Pulsantes

Nesta explanação os modelos de neurônios estão divididos em três categorias: modelo de disparo por limiar, modelo baseado em condutância e modelo compartimental [GK02b]. Uma quarta categoria, os modelos de taxa de disparo, foi também considerada [Maa97, Ger99], embora não se trate de neurônio pulsante.

Os modelos de disparo por limiar são baseados na soma temporal de todas as contribuições ao potencial de membrana \( u(t) \) recebidas dos neurônios pré-sinápticos. Se essa soma ultrapassa um limiar \( \theta \), então o neurônio pós-sináptico dispara. Exemplos desta categoria são os modelos Integra e Dispara – I&F e os modelos de Resposta de Pulso – MRP [Ger01, GK02b].

![Diagrama de neurônio pulsante](image)

Figura 3.1: Neurônio integra-e-dispara (de [MB98]).

O modelo Integra e Dispara é composto pelo circuito RC mostrado na figura 3.1 e é descrito pela equação (3.1).

\[
I(t) = \frac{u(t)}{R} + C \frac{du}{dt}
\]

(3.1)
Fazendo-se a constante de tempo \( \tau_m = RC \) pode-se reescrever 3.1 na forma da equação (3.2),

\[
\tau_m \frac{du}{dt} = -u(t) + RI(t)
\]

(3.2)

onde \( \tau_m \) é a constante de tempo de membrana. O potencial de recuperação é definido pela equação (3.3).

\[
\lim_{t \to t^+_f; t > t_f} u(t) = u_{rest}
\]

(3.3)

O comportamento do modelo é então regido pelas equações (3.2) e (3.3).

Considerando uma corrente de entrada constante \( I_0 \), um potencial de repouso \( u_{rest} = 0 \) e integrando (3.2) com as condições iniciais \( u(t^{(i)}) = u_{rest} = 0 \), obtemos a equação (3.4).

\[
u(t) = RI_0 \left[ 1 - e \left( -\frac{t-t^{(i)}}{\tau_m} \right) \right]
\]

(3.4)

Existem inúmeras variações deste modelo [Izh04, Coo03, Rul02, Izh01].

No modelo MRP o estado de um neurônio \( j \) é descrito por uma única variável de estado \( u_j(t) \). Se \( u_j(t) \) cruza desde abaixo \( \left( \frac{du}{dt} > 0 \right) \) um limiar \( \theta \) em um instante \( t^{(f)}_j \), então um pulso é gerado. O valor da variável de estado \( u_j(t) \) é influenciado por dois processos. O primeiro é a contribuição dos neurônios pré-sinápticos \( i \in \Gamma_j \), que modela a resposta aos neurônios pré-sinápticos e é dada pela equação (3.5).

\[
h(t) = \sum_{i \in \Gamma_j} \sum_{j^{(f)}_i \in \Gamma_j} w_{ij} \varepsilon_{ij} \left( t - \hat{\tau}_j, t - t^{(f)}_j - \Delta_{ij}^{ax} \right) + I_{j}^{ext}
\]

(3.5)

Onde \( w_{ij} \) é a eficácia sináptica, \( I_{j}^{ext} \) é uma excitação externa e \( \varepsilon_{ij} \) é o PPS, mostrado na figura 2.4 e equações (2.2) e (2.3). Para simplificar será usado \( s = t - t^{(f)}_j - \Delta_{ij}^{ax} \).

O segundo processo é a recuperação de \( u_j(t) \) até que \( u_{rest} \) seja alcançado (período refratário). Este termo é modelado por \( \eta_j(t - \hat{\tau}_j) \), que é uma função do último instante de disparo de \( j \) (\( \hat{\tau}_j \)) [MB98].

A expressão para o último termo de (3.5) é dada pela equação (3.6).

\[
\int_{0}^{\infty} k(t - \hat{\tau}_j, s) I_{j}^{ext}(t - s) ds
\]

(3.6)
O potencial pós-sináptico total é então

\[ h(t|\tilde{t}_j) = \sum_I w_{ij} \sum_{j' \in I} c_{ij}(t - \tilde{t}_j, s) + \int_0^\infty \kappa(t - \tilde{t}_j, s) I_{\text{ext}}(t - s) ds \]  

(3.7)

Os modelos baseados em condutância são muito complexos e são baseados no comportamento dos canais de íons. Um exemplo desta classe de modelos é o modelo de Hodgkin e Huxley [HH52].

Figura 3.2: Diagrama esquemático para o modelo de Hodgkin-Huxley (de [MB98]).

Tomando a figura 3.2 como referência, considerando que a corrente total deve ser igual à soma das correntes nos ramos, temos

\[ I(t) = I_c(t) + \sum_{ch} I_{ch}(t) \]  

(3.8)

uma vez que \( I_c = C \frac{du}{dt} \), então

\[ C \frac{du}{dt} = - \sum_{ch} I_{ch} + I(t) \]  

(3.9)

O potencial de membrana é dado por \( u(t) \) e \( \sum_{ch} I_{ch} \) é a soma das correntes iônicas atravessando a membrana da célula.

Considerando que um canal está aberto ou fechado, três outras variáveis são necessárias [HH52], produzindo a equação (3.10) para a soma das três correntes. Os parâmetros estão listados na tabela 3.
Tabela 3.1: Parâmetros da equação de Hodgkin-Huxley. As unidades usadas são \( mS/cm^2 \) para condutâncias, \( mV \) para o potencial de equilíbrio e \( \mu F \) para capacitâncias.

<table>
<thead>
<tr>
<th>( x )</th>
<th>( E_x(mV) )</th>
<th>( g_x(mS/cm^2) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Na</td>
<td>50</td>
<td>120</td>
</tr>
<tr>
<td>K</td>
<td>-77</td>
<td>36</td>
</tr>
<tr>
<td>L</td>
<td>-54.4</td>
<td>0.3</td>
</tr>
</tbody>
</table>

Tabela 3.2: Parâmetros das equações para \( a_x \) e \( \beta_x \).

<table>
<thead>
<tr>
<th>( x )</th>
<th>( a_x(u/mV) )</th>
<th>( \beta_x(u/mV) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m )</td>
<td>((2.5 - 0.1u)/[\exp(2.5 - 0.1u) - 1])</td>
<td>(4\exp(-u/18))</td>
</tr>
<tr>
<td>( h )</td>
<td>(0.07\exp(-u/20))</td>
<td>([1/\exp(3 - 0.1u) + 1])</td>
</tr>
<tr>
<td>( n )</td>
<td>((0.1 - 0.01u)/[\exp(1 - 0.1u) - 1])</td>
<td>(0.125\exp(-u/80))</td>
</tr>
</tbody>
</table>

\[
\sum_{ch} I_{ch} = g_{Na}m^3h(u - E_{Na}) + g_Kn^4(u - E_K) + g_L(u - E_L) \tag{3.10}
\]

Se os canais estão sempre aberto, as correntes podem ser definidas simplesmente por \( I_x = g_xu \) [Koc99]. A equação (3.9) pode então ser escrita como

\[
C \frac{d\phi}{dt} = -[g_{Na}m^3h(u - E_{Na}) + g_Kn^4(u - E_K) + g_L(u - E_L)] + I(t) \tag{3.11}
\]

As três novas variáveis são dadas pelas seguintes equações diferenciais

\[
\frac{dm}{dt} = a_m(u)(1 - m) - \beta_m(u)m \ 
\frac{dn}{dt} = a_n(u)(1 - n) - \beta_n(u)n \tag{3.12}
\frac{dh}{dt} = a_h(u)(1 - h) - \beta_h(u)h
\]

As funções \( a \) and \( \beta \) são empíricas e estão mostradas na tabela 3. As equações (3.11) e (3.12) definem o modelo de Hodgkin-Huxley. A figura 3.3 mostra as funções de equilíbrio e as constantes.
de tempo para as variáveis \( m, n, \) e \( h. \) Os valores para \( m_0, n_0, h_0, \tau_m, \tau_n \) e \( \tau_h \) são dados pelas equações (3.13).

\[
x_0 = \frac{\alpha_x(u)}{[\alpha_x(u) + \beta_x(u)]}
\]

\[
\tau_x = \frac{1}{[\alpha_x(u) + \beta_x(u)]}
\]

(3.13)

onde \( x \) substitui \( m, n \) or \( h. \)

Figura 3.3: **Eq.**: ativação de regime permanente (\( m \) para sódio e \( n \) para potássio) e inativação (\( h \) para sódio). **Dir.**: constantes de tempo em função do potencial de membrana (de [Koc99, GK02b]).

A grande complexidade dos neurônios reais é melhor capturada pelos modelos compartimentais, que levam em conta a estrutura espacial da árvore dendrítica e também modelam a transmissão sináptica em um maior nível de detalhe.

A ideia básica por trás dos modelos compartimentais é a divisão dos componentes em partes menores e uniformes. Cada componente é então modelado por equações descrevendo o circuito elétrico equivalente, mostrado na figura 3.4, onde \( u \) é o potencial de membrana, através da capacitância da membrana \( C. \) A equação diferencial resultante é dada pela equação (3.14).

\[
C \frac{du}{dt} = (E - u) G + \sum_{ch} (E_{ch} - u) G_{ch} + \frac{(u' - u)}{R_a} + \frac{(u - u)}{R_g} + I
\]

(3.14)

Os modelos de taxa de disparo, também conhecidos como unidades sigmoidais, são os modelos mais empregados na construção de redes neurais artificiais para aplicações de engenharia. Estes modelos foram inicialmente propostos em [MP43] e um exemplo mais atual é o Perceptron [Ros58, MP88].
Figura 3.4: Um circuito equivalente "genérico" de um modelo compartimento neural (de [BB03]).

A resposta $v_j$ de um neurônio $j$ é também uma taxa e depende de uma função sigmoidal. A entrada para esta função é a soma de todas as excitações aplicadas em $j$, isto é, todas as taxas de saídas dos neurônios pré-sinápticos $i \in \Gamma_j$:

$$v_j = g \left( \sum_{i \in \Gamma_j} w_{ij} v_i \right)$$  \hspace{1cm} (3.15)

onde $w_{ij}$ é o peso sináptico entre os neurônios $i$ e $j$. A equação (3.15) é o ponto de partida para a teoria das tradicionais redes neurais artificiais [Hay99, BLdC00].
Capítulo 4

Neurodinâmica

Neste capítulo são apresentados exemplo de aplicação dos modelos para simular o comportamento dinâmico de neurônios reais [And04], conforme descrito nos capítulos anteriores. O comportamento dinâmico de sistemas neurais é usualmente descrito por equações diferenciais multidimensionais, cujos resultados são difíceis de visualizar e, mais ainda, de analisar [Kin91]. O uso de modelos simplificados, bidimensionais, como os de Morris-Lecar [ML81] ou FitzHugh-Nagumo [Fit61, NAY62], facilitam essa tarefa.

Um exemplo de aplicação dos modelos simplificados é a simulação da histerese, um importante fenômeno não-linear observado na presença de uma bifurcação subcrítica de Hopf [ASY00, HI01]. Sua existência em sistema neurais reais foi demonstrada em um axônio gigante de lula [GLR80].

\[
C \frac{du}{dt} = -(17.81 + 47.7u + 32.63u^2)(u - 0.55) - 26.0w(u + 0.92) + I
\]

\[
\frac{dw}{dt} = \frac{1}{\tau}(-w + 1.35u + 1.03)
\]

(4.1)

O experimento é aqui repetido usando a equação (4.1) [Wil99]. Uma ilustração dos resultados está mostrada na figura 4.1.

Desta breve discussão verifica-se que a aplicação de modelos mais elaborados tem se restrigido a simulações de sistemas neurais reais, não sendo observada seu uso em aplicações mais práticas, talvez devido à pouca tratabilidade dos modelos mais complexos.
Figura 4.1: Histerese no axônio de lula produzido em resposta a uma rampa triangular de corrente, com valores variando de \( I = 0 \) a \( I = 0.25 \mu A \) (mostrado aqui fora de escala, apenas para referência). Os parâmetros na equação (4.1) foram \( C = 0.8 \mu F/cm^2 \) e \( \tau = 1.9 \)
Capítulo 5

Uma Aplicação Prática Usando RNP

Uma vez definidos os conceitos princípios básicos e conhecidos os principais modelos, apresentamos aqui uma aplicação prática mais concreta de uma RNN. A aplicação de RNN em tarefas de clusterização não é recente [Hop95] e pode ser encontrada na literatura científica [SW99, Jin04, KIKY03]. Os exemplos discutidos aqui foram extraídos principalmente de dois trabalhos [Ruf98, BPK02] e sua implementação foi desenvolvida em scripts MATLAB [HL03].

A primeira questão é definir um método para traduzir grandezas analógicas em uma codificação temporal [HMR02], de forma que uma RNN possa usá-las. Este tema é também muito discutido na literatura [SKdRVSB98, AT02, YS99, AA99, Koc99, PP95], mas ainda encontra-se sem uma solução definitiva [RWdRVSB97]. Neste trabalho será usada uma codificação baseada em população [Sni96, dKvdV01], como mostrado na figura 5.1.

Para o aprendizado foi empregado um método Hebbiano por reforço não-supervisionado [Heb49], implementado através de um algoritmo do tipo winner-takes-all [BLdC00, Hay99], cuja aplicação prática é discutida em [Ruf98, BPK02, dQ02] e uma abordagem mais teórica é apresentada em [GK02a, BSS93].

A adaptação dos pesos foi realizada conforme a equação equação (5.1), onde \( w_{ij} \) são os pesos sinápticos e \( \eta \) é a taxa de aprendizado. A função \( L(\Delta t_{ij}) \) é chamada de função de aprendizado [KGvH99a, KGvH99b] e é uma função do intervalo de tempo entre o disparo dos neurônios pré- e pós-sinápticos \( \Delta t_{ij} \).

\[
\Delta w_{ij} = \eta L(\Delta t_{ij}) \tag{5.1}
\]

A função de aprendizado usada é definida pela equação (5.2) [GK02a, KGvH99a, KGvH99b, LvH01], e está mostrada na figura 5.2.
Figura 5.1: Variáveis contínuas de entrada codificadas por meio de campos receptivos locais.

\[
L(\Delta t) = (1 + \beta) e^{\frac{(\Delta t - \alpha)^2}{2(\kappa - 1) - \beta}}
\]

(5.2)

where

\[
\kappa = 1 - \frac{\nu^2}{2 \ln \frac{\beta}{1 + \beta}}
\]

A arquitetura adotada é similar à arquitetura de redes neurais artificiais convencionais, com os neurônios da camada precedente totalmente conectados aos neurônios da próxima camada. Uma diferença observada é que os neurônios de saída são interconectados por sinapses inibitórias, de forma a implementar o algoritmo winner-takes-all. Outra diferença notável é o emprego de sinapses múltiplas [WZR98], como mostrado na figura 5.3.

Com esta configuração, com cada sinapse múltipla tendo \( s \) sub-sinapses, o PPS resultante entre o neurônios \( i \) e \( j \) é dado pela equação (5.3).

\[
u_{ij}(t) = \sum_{k=1}^{s} w_{ij}^k \epsilon(t - t_i - d^k)
\]

(5.3)
Figura 5.2: Função de aprendizado gaussiana, com $\alpha = -3$, $\beta = 0.2$, e $\nu = 4$.

onde $d^k$ tem o mesmo efeito que um atraso axonal. A contribuição total é dada pela equação (5.4).

$$u_j(t) = \sum_{i \in I_j} \sum_{k=1}^{s} w_{j,k} \epsilon(t - t_i - d^k)$$  \hspace{1cm} (5.4)

O modelo neural usado foi o $SRM_0$ [GK02b], com um PPS estritamente excitatório, modelado pela equação (2.3), com $\tau = 3\text{ms}$. O primeiro exemplo apresentado mostra a correta separação entre cinco grupos gerados aleatoriamente (figura 5.4).
Figura 5.3: Sinapse Múltipla. O PPSE final é a soma de todos os potenciais produzidos pelas sub-sinapses.

O segundo exemplo mostra a separação das três cores básicas em quatro grupos, em uma imagem padrão RGB, de 256 x 256 pixels (figura 5.5).

Como uma contribuição deste trabalho é proposta uma nova abordagem para a realização das tarefas de clusterização, com a utilização de sinapses simples. Esta nova abordagem é demonstrada com a utilização de um PPS produzido pela sinapse simples de forma que este seja equivalente ao PPS produzido por uma sinapse múltipla, como mostrado na figura 5.6.

Os resultados obtidos e a complexidade do problema indicam ser ainda necessário submeter o método proposto a uma análise matemática bastante mais profunda do que seria abrangido pelo escopo deste trabalho, ficando esta tarefa para um possível trabalho futuro.
Figura 5.4: Cinco grupos corretamente separados por uma RNP com 8 neurônios codificando cada dimensão. Foram usadas 13 sub-sinapses com atrasos variando de 0 a 12ms e uma taxa de aprendizado $\eta = 0.35$. 
Figura 5.5: Resultados da clusterização das cores de uma imagem. Em sentido horário: imagem original, clusterização com o algoritmo k-means, clusterização usando SOM (Matlab) e clusterização com RNP.
Figura 5.6: Dois PPSE (linha tracejada) equivalentes ao produzido por um conjunto de sinapses múltiplas (linha sólida). Equação (2.3) foi usada em ambos os casos.
Capítulo 6

Conclusões

Como era nosso objetivo, apresentamos os fundamentos de neurônios pulsantes e enfatizamos sua importância na construção das redes neurais artificiais em futuro próximo. Para isso foram discutidos os componentes básicos dos sistemas neurais biológicos e, em seguida, apresentamos os modelos matemáticos que permitem simular estes componentes e sistemas.

Após uma discussão mais teórica, foram introduzidos exemplos práticos de aplicação dos modelos biologicamente inspirados para demonstrar o comportamento dinâmico dos sistemas neurais. Além disso, os modelos foram também utilizados para realizar tarefas de clusterização.

Além da revisão extensiva e dos exemplos práticos, foi apresentada uma nova abordagem para a implementação de uma RNP de menor custo computacional. Embora o campo aqui abordado seja muito vasto, ficam como propostas para trabalhos futuros duas linhas de pesquisas, a saber, codificação neural e a nova abordagem proposta no capítulo anterior.
Referências Bibliográficas


A Review of Spiking Neuron Models and Applications

by

Roberto Coelho de Berrêdo

B.S., Pontifical Catholic University of Minas Gerais, Brazil, 1978.

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
Master of Science

in the
Post-Graduation Program in Electrical Engineering

we accept this dissertation as conforming
to the required standard

Universidade Federal de Minas Gerais

November 2005

© Roberto Coelho de Berrêdo, 2005
Abstract

Although the nervous system and nervous cells have been studied since ancient times, almost all the important advances on the knowledge base took place along the past century, when neuroscience evolved and developed more complex and detailed neuron models. However, engineering applications were – and still are – practically limited to the basic neuron models, like the MCP [MP43] and its variations. In traditional artificial neural networks, the neuron behavior is described only in terms of firing rate, while most real neurons, commonly known as spiking neurons, transmit information by means of pulses, also called action potentials or spikes. From these considerations a major question raises immediately: if we were able to build powerful applications in all fields of engineering using these simple models, what would it be possible to do with the more complex models? This question is the fundamental motivation of the present work.

Given the importance of more realistic neuron models, our main objective is to present a general and comprehensive overview of spiking neurons, ranging from biological neuron features to examples of practical applications. It will be also demonstrated how analog information can be coded by precisely timed spikes, emitted by different neurons, and how this coded information can be processed to produce useful results [Maa97]. The aim of the present work is therefore to highlight what we believe will be one of the main components of the future computing machines: the spiking neuron.

As a contribution, besides the review itself, we present also a novel approach to the spiking neuron network architecture used in practical applications [Ruf98, BPK02], replacing the multiple synapse construction with single synapses, and thus enhancing the computational efficiency of the network.
## Contents

Abstract ............................................. ii

Contents .......................................... iii

List of Tables ..................................... vi

List of Figures .................................... vii

Dedicatória ....................................... ix

1 Introduction ..................................... 1
   1.1 Background ................................... 1
   1.2 Objective and Organization of the Work ........... 2

2 Biological Neurons and Neuronal Features .... 5
   2.1 Introduction ................................... 5
   2.2 A Generic Spiking Neuron ....................... 5
       2.2.1 The Nervous Cell Membrane ................... 7
       2.2.2 Synapses ..................................... 7
       2.2.3 The Postsynaptic Potential – PSP ............ 10
       2.2.4 The Spike After Potential – SAP ............. 11
   2.3 The Equilibrium Potential ....................... 12
   2.4 Neurons Main Characteristics and Features ........ 14
   2.5 Conclusions .................................... 17

3 Spiking Neuron Models ........................... 18
   3.1 Introduction ................................... 18
   3.2 Threshold-Fire Models ......................... 18
       3.2.1 Integrate-and-Fire Model – I&F ............... 19
       3.2.2 Spike Response Model – SRM ................ 23
List of Tables

2.1 Extracellular and intracellular ion concentrations ........................................ 14
3.1 Parameters of the Hodgkin-Huxley equation .................................................... 27
3.2 Parameters of the equations for $\alpha_x$ and $\beta_x$ ........................................ 27
5.1 Main parameters in the three examples of this chapter. .................................... 54
# List of Figures

1.1 Separation of ten clusters .................................................. 4
2.1 Generic neuron diagram .................................................... 6
2.2 Synapse diagram ............................................................ 7
2.3 EPSP and IPSP .............................................................. 10
2.4 Example of an EPSP ........................................................ 12
2.5 The SRM spike .............................................................. 13
2.6 Neuro-computational features ............................................ 16
3.1 Integrate-and-fire neuron .................................................. 19
3.2 Membrane potential ........................................................ 20
3.3 Membrane potential of a integrate-and-fire neuron ............... 21
3.4 Membrane potential of a quadratic integrate-and-fire neuron . 22
3.5 Dynamic threshold ........................................................ 24
3.6 The SRM₀ model ........................................................... 25
3.7 The Hodgkin-Huxley model .............................................. 26
3.8 Action potential of the HH model ...................................... 26
3.9 Parameters of the HH model ............................................. 29
3.10 FitzHugh-Nagumo model ............................................... 29
3.11 Izhikevitch model ......................................................... 32
3.12 Equivalent circuit of a compartmental model ..................... 33
3.13 Naka-Rushton equation plot ........................................... 35
4.1 Limit cycle ................................................................. 37
4.2 Hysteresis in the squid axon ............................................. 38
4.3 Slow rate firing of a class 1 excitable neuron ......................... 39
4.4 Phase plane and nullcline for a human neuron simulation ....... 40
4.5 Gain function for Class 1 and 2 neurons ............................. 40
4.6 Chaotic spike train ........................................... 41
5.1 Simple temporal encoding ........................................ 44
5.2 SNN example .................................................. 45
5.3 Encoding by receptive fields .................................... 46
5.4 Encoding spikes ................................................ 47
5.5 Two types of receptive fields ................................... 48
5.6 Learning function .............................................. 49
5.7 Multiple synapse ............................................... 50
5.8 EPSP of 13 sub-synapses ....................................... 51
5.9 Weight evolution ................................................ 52
5.10 SNN results .................................................... 53
5.11 Image clustering ................................................. 55
5.12 Single-synapse EPSP ........................................... 56
5.13 Alternative learning function ................................. 58
5.14 Relationship between weights and time constants .......... 59
To my family, professors and friends, for their support and patience.
Chapter 1

Introduction

1.1 Background

Although the nervous system and nervous cells have been studied since ancient times, almost all the important advances on the knowledge base took place along the past century, and were founded on five experimental disciplines: anatomy, embryology, physiology, pharmacology, and psychology [KSJ00]. Then, beginning in the middle of the past century, engineering was added as a sixth discipline and, reciprocally, neuroscience was also adopted by engineering, ensuing all the development of computational intelligence and making this subject a rather interdisciplinary one.

However, as neuroscience evolved and developed more complex and detailed neuron models, engineering applications were – and still are – practically limited to the basic neuron models, like the MCP [MP43] and its variations. In traditional artificial neural networks, the neuron behavior is described only in terms of firing rate, while most real neurons, commonly known as spiking neurons, transmit information by pulses, also called action potentials or spikes. From these considerations a major question raises immediately: if we were able to build powerfull applications in all fields of engineering using these simple models, what would it be possible to do with the more complex models?

Moreover, in the last ten years or so all the main questions formerly addressed by neuroscientists only to cellular biology began to be effectively explored in the molecular level. The research in molecular biology contributed to the knowledge about ion channels and receptors, two important elements in neural signaling, making it possible to describe the first molecular structure of an ionic channel [KSJ00]. This enhanced capacity of neuronal modelling came to shed some light into questions like: How do the nervous cells communicate among them? How is this communication modified by experience? How different interconnection patterns originate different perceptions and motor actions?
From the engineering point of view, it is clear that the answers for all these questions will only be possible with a deeper comprehension of the biological neuron and how they can do fast and reliable computation [MB98]. This apparent difficulty should rather be regarded as an opportunity to use spike-timing as an additional variable in the information processing by neural networks [Boh04]. Although much progress has been achieved in the last two decades, there are a few fundamental questions still not completely solved, like how do real neurons transmit information or how to use the spike timing efficiently to process information [Nat96, GK02b].

As a matter of fact, realistic and complex neural models have been used since long to simulate real networks, but always with emphasis on the spike rate. It was also only in the last ten years, however, that we have witnessed a shift on the emphasis in the artificial neural network community toward practical spiking neural networks - SNN [TH86, Hop88, Hop95, Nat96]. This shift was motivated by recent biological discoveries and by the increasing available computer power, so that many researchers began considering pulse-coupled neural networks with spike-timing as an essential component in information processing by neural networks [Maa97, Ger01], artificial or not.

Independent of the choice of the point of view (molecular, cellular, systemic, behavioral, or cognitive), the human nervous system is a masterpiece of biological machinery, worth to be better studied and employed as a powerfull engineering tool. All its accomplishments are more than sufficient motivation for willing to understand how each of its many components works. Not to mention that the debilitating and costly effects of neurological and psychiatric disease add a further sense of urgency to this quest [PAF+04].

It is likely that next great change in computer science and information technology may come from mimicking the way by which biological neural systems process information. To accomplish this, engineers and computer scientists will have draw on expertise in subjects not usually associated with their fields, including organic chemistry, molecular biology, bioengineering, and smart materials. In a not so distant future we will be possibly using proteins and other molecules for information-processing, molecular recognition, computation in nonlinear media, computers based on physical reaction-diffusion systems found in chemical media, DNA computing, bioelectronics and protein-based optical computing, and biosensors [SARC03].

1.2 Objective and Organization of the Work

Given the importance of more realistic neuron models, as exposed in the previous section, the main objective of this work is to present a general and comprehensive overview of spiking neurons, ranging from biological neuron features to examples of practical applications. It will be also
demonstrated how analog information can be coded by precisely timed spikes, emitted by different neurons, and how this coded information can be processed to produce useful results, as demonstrated theoretically in [Maa97]. The aim of the present work is therefore to highlight what we believe will be one of the main components of the future computing machines: the spiking neuron. In this work we intend to make a review on spiking neurons, presenting a comprehensive scenario of the foundations and a few application examples as well.

The present knowledge of the nervous system has reached an enormous level of detail, making it impossible even to summarize its main components in only one chapter. Therefore, in Chapter 2 we limited ourselves to mention only those main components with a brief description of its basic properties, including ion channels, synapses, dendrites and axons, as well as the electrical features like excitatory (inhibitory) postsynaptic potential and the spike. The purpose of this chapter is to provide enough information about the nervous system components, preparing a basis for the next chapter, when we will discuss the features and characteristics we must take into account when designing a neuron model. For the sake of objectiveness we do not included those components with less significant functions in the information processing in a nervous system (e.g. the glial cells), although they may play a significant role in the brain structure.

Since the works of Santiago Ramon y Cajal [yC03] and Camillo Golgi, a vast number of theoretical neuron models have been created, with a modern phase beginning with the work of Hodgkin and Huxley [HH52]. All these models range from special purpose to generic models, aiming to emulate one or various features of the nervous systems, most of which are mentioned in the previous chapter. In Chapter 3 we concisely present some of the most used theoretical neuron models, from the simple ones, like the integrate-and-fire, to the more complex ones, like the compartmental models. In real nervous systems there is a wide range of neuron types, each one assigned to do a specific function, making it virtually impossible to create a model that meets all the requirements.

Historically, much of the research effort to comprehend the neural mechanisms involved in information processing in the brain has been spent with neuronal circuits and synaptic organization, largely neglecting the electrophysical properties of the neurons. In Chapter 4, some of the theoretical models presented in Chapter 3 are implemented, giving some examples of the application of the models to simulate real neurons. These examples demonstrate the existence of a relationship between electrophysiology, bifurcations, and computational properties of neurons, showing also the foundations of the dynamical behavior of neural systems.

In Chapter 5 we present an example of a practical application using spiking neurons and temporal coding to process information, building a spiking neural network – SNN to perform a clustering task, as shown in figure 1.1. The input is encoded by means of receptive fields, as
Figure 1.1: The SNN at left was capable of correctly separate ten clusters. The lateral connections linking the output neurons are strong inhibitory synapses, disabling all other neurons to fire after the first neuron has fired, thus implementing the winner-takes-all process. Each dimension of the input was coded by a receptive field with 12 neurons.

described in [Ruf98] and [BPK02]. The delay and weight adaptation uses a multiple synapse approach, dividing each synapse into subsynapses, each one with a different fixed delay. The delay selection is then performed by a Hebbian reinforcement learning algorithm, also keeping resemblance with biological neural networks. As a contribution of this work, we propose in this chapter an alternative architecture to do the same task with less computing effort. This new approach is based on the use of an EPSP produced by a single synapse that is equivalent to the EPSP produced by the multiple synapse architecture of the original method.

Finally, in Appendix A we listed some of the Matlab scripts [HL03] used in the development of Chapter 5, and in Appendix B we listed some interesting internet sites, which were very helpful to accomplish this work and where a lot of usefull material can be found.
Chapter 2

Biological Neurons and Neuronal Features

2.1 Introduction

The aim of this chapter is to present some elementary notions of neuroscience, beginning with a generic phenomenological model of the biological neuron and its basic concepts, like action potentials, postsynaptic potentials, firing thresholds and refractoriness. This will serve as the starting point to the following chapter, where the various theoretical models used in computational neuroscience will be discussed.

The human brain is made of hundreds of billions of cells and a full description of this system, as far as it has been uncovered, would take several volumes of written material. Due to this high complexity, it is beyond the scope of this work to present a complete introduction to this subject. Therefore, we limit ourselves to a highly selective and simplistic biological background to provide the minimum amount of information necessary to the development of this work.

Even the basic nervous cell, the neuron, has many different forms and is spread over several anatomically different structures, like brain-stem, cerebellum, and cortex. In the following sections we limit ourselves to the presentation of the basic concepts of a generic neuron model and the most important types of behaviors and features of spiking neurons.

2.2 A Generic Spiking Neuron

Throughout the past hundred years, researchers have been uncovering an increasingly complex brain structure. The elementary processing units in the brain are the neurons, which are connected to each other in an intricate pattern and can occur in many shapes and sizes. As stated above, the neuron described here is very simplified and its purpose is to serve as a 'template' neuron for
further development. Our generic neuron has four functionally distinct parts, called dendritic tree, soma, axon and synapse, like the diagram shown in figure 2.1 [yC03]. Roughly speaking, signals from other neurons are collected by the dendrites (input device) and are transmitted to the soma (central processing unit). If the total excitation caused by the input is sufficient, i.e., above a threshold, an output signal (action potential, or spike) is emitted and propagated along the axon (output device) and its branches to other neurons. It is in the transition zone between the soma and the axon, the axon hillock, where the essential non-linear processing step occurs [KS]00.

![Neuron Diagram]

Figure 2.1: A simplified generic neuron diagram (based on [yC03]).

It is common to refer to a sending neuron as the presynaptic neuron and to the receiving neuron as a postsynaptic neuron. A neuron in vertebrate cortex often makes connections to more than $10^5$ postsynaptic neurons [RWdRVSB97]. Many of its axonal branches end in the direct neighborhood of the neuron, but the axon can also stretch over several millimeters and connect to neurons in other areas of the brain.

The neuronal signals can be observed by placing a fine electrode close to the soma or axon of a neuron. The voltage trace in a typical recording shows a sequence of short pulses, called action potentials or spikes. A chain of pulses emitted by a single neuron is usually called a spike train— a sequence of stereotyped events which occur at regular or irregular intervals. The duration of an action potential is typically in the range of $1 - 2ms$ with an amplitude of about $100mV$ [GK02b, PAF+04], as showed in figure 2.3. Since all spikes of a given neuron look alike, the form of the action potential does not carry any information. Rather, it is the number and the timing of spikes which matter [MB98].
2.2.1 The Nervous Cell Membrane

Neurons employ several different types of electrical signals to encode and transfer information. These signals are caused by responses to stimuli and are generated by means of elaborated mechanisms, usually based on the flow of ions across the plasma membranes of nervous cells [KSJ00]. The electrical potential across the membrane, called membrane potential, is generated by different concentrations of specific ions inside and outside of nerve cells. Each membrane has its specific resting potential (typically -40 to -90mV) [PAF+04]. The ion concentration (and the membrane potential) gradient is regulated by the active transporters and ion channels.

The active transporters, also known as ion pumps, are proteins that actively move ions into or out of cells against their concentration gradients. The ion channels are proteins that allow only certain kinds of ions to cross the membrane in the direction of their ion concentration gradients, and thus characterizing the selective permeability of the membrane to each type of ion. These two mechanisms basically work against each other, generating membrane resting potentials, action potentials and synaptic potentials.

![Diagram of a synapse with its vesicles, neurotransmitters and neurotransmitters](image)

Figure 2.2: Diagram of a synapse with its vesicles, neurotransmitters and neuroreceptors.

2.2.2 Synapses

Information from one neuron flows to another neuron across a small gap, called synapse, separating an axonal branch and the dendrite (or the soma) of a receiving neuron. The most common type of synapse in the vertebrate brain is the chemical synapse [PAF+04, KSJ00], which act through
biochemical processes, neurotransmitters and receptors to change the membrane potential of the postsynaptic neuron. In chemical synapses, an action potential cannot cross the synaptic cleft between neurons. Instead, the nerve impulse is carried by chemicals called neurotransmitters, which are made by the cell that is sending the impulse (the presynaptic neuron) and stored in synaptic vesicles at the end of the axon. The cell that is receiving the nerve impulse (the postsynaptic neuron) has chemical-gated ion channels in its membrane, called neuroreceptors. These have specific binding sites for the neurotransmitters. As shown in figure 2.2 chemical synapse consists of [PAF+04]:

- a presynaptic ending that contains neurotransmitters, mitochondria, and other cell organelles (where neurotransmitters are synthesized),
- a postsynaptic ending that contains receptor sites for neurotransmitters and has neuroreceptors in the membrane,
- a synaptic cleft or space between the presynaptic and postsynaptic endings. It is about 20 nm wide.

The action potential transmission process can be described by the following steps [KS00]:

a. Transmitter is synthesized and then stored in vesicles.

b. An action potential reaches the presynaptic terminal.

c. At the end of the presynaptic neuron there are voltage-gated calcium channels. When an action potential reaches the synapse, depolarizing the presynaptic terminal, these channels open, causing calcium ions ($Ca^{2+}$) to flow into the cell.

d. These calcium ions ($Ca^{2+}$) cause the synaptic vesicles to fuse with the cell membrane, releasing their contents (the neurotransmitter chemicals) via exocytosis.

e. The neurotransmitters are released into the synaptic cleft and diffuse to the postsynaptic terminal across the synaptic cleft.

f. The neurotransmitter binds to the neuroreceptors in the postsynaptic membrane, causing its channels to open (or close). This causes a depolarization (EPSP) or hyperpolarization (IPSP) of the postsynaptic cell membrane, which may initiate an action potential, if the threshold is reached. See figure 2.5.

g. Retrieval of vesicular membrane from plasma membrane. The neurotransmitter is broken down by a specific enzyme in the synaptic cleft (e.g., the enzyme acetylcholinesterase breaks
down the neurotransmitter acetylcholine). The breakdown products are absorbed by the presynaptic neuron by endocytosis and used to re-synthesize more neurotransmitters, using energy from the mitochondria. This stops the synapse being permanently on.

Apart from chemical synapses, neurons in the mammalian nervous systems can also be coupled by other different types of synapses. The human nervous system uses a number of different neurotransmitters and neuroreceptors, and they do not all work in the same way. We can group synapses into 5 types [PAF+04, KSJ00]:

**Excitatory Ion Channel Synapses:** These synapses have neuroreceptors that are sodium channels. When the channels open, positive ions flow in, causing a local depolarization and making an action potential more likely. This was the kind of synapse described above. Typical neurotransmitters are acetylcholine, glutamate or aspartate. Neureceptors are Na+ channels. When Na+ channels open, local depolarization occurs, if threshold is reached, then action potential is initiated.

**Inhibitory Ion Channel Synapses:** These synapses have neuroreceptors that are chloride channels. When the channels open, negative ions Cl⁻ flow in causing a local hyperpolarization and making an action potential less likely. So, with these synapses an impulse in one neuron can inhibit an impulse in the next. Typical neurotransmitters are glycine or GABA (γ-aminobutyric acid).

**Non Channel Synapses:** These synapses have neuroreceptors that are not channels at all, but instead are membrane-bound enzymes. When activated by the neurotransmitter, they catalyze the production of a ‘chemical messenger’ inside the cell, which, in turn, can affect many aspects of the cell’s metabolism. In particular, they can alter the number and sensitivity of the ion channel receptors in the same cell. These synapses are involved in slow and long-lasting responses like learning and memory. Typical neurotransmitters are adrenaline, noradrenaline (NB adrenaline is also called epinephrine), dopamine, serotonin, endorphin, angiotensin, and acetylcholine.

**Neuromuscular Junctions:** These are the synapses formed between motor neurons and muscle cells. They always use the neurotransmitter acetylcholine, and are always excitatory. Motor neurons also form specialized synapses with secretory cells. Neuroreceptors are membrane-bound enzymes. When activated, they catalyze the ‘chemical messenger’, which in turn can affect the sensitivity of the ion channel receptors in the cell synapses formed between motor neurons and muscle cells.
**Electrical Synapses:** In these synapses, the membranes of the two cells actually touch, and they share proteins. This allows the action potential to pass directly from one membrane to the next. They are very fast, but are quite rare, found only in the heart and in the eye.

### 2.2.3 The Postsynaptic Potential – PSP

The potential difference between the interior of the cell (soma) and its surroundings is called the membrane potential. This potential is directly affected by the postsynaptic potentials - PSPs generated by the spikes received from presynaptic neurons. If the membrane potential reaches a threshold, an action potential (spike) is triggered and sent out through the axon and its branches to the postsynaptic neurons. If the postsynaptic potential is positive, it is said to be excitatory (EPSP) and if the change is negative, the synapse is inhibitory (IPSP). See figure 2.3. The process of spike transmission through the axon has an associated delay, called the axonal delay [PAF⁺04, KSJ00].

![Graph showing EPSP and IPSP](image)

Figure 2.3: Examples of action potential (left) and EPSP and IPSP (right).

At rest, the cell membrane already has a strong negative polarization of about $-65mV$ [Wil99]. An input at an excitatory synapse reduces the negative polarization of the membrane and is therefore called depolarizing. Likewise, an inhibitory synapse increases the negative polarization of the membrane even further and is called hyperpolarizing. Single EPSPs have amplitudes in the range of $1mV$ and the threshold for spike initiation is about $20 – 30mV$ [GK02b] above the resting potential.

At this point we need to define some notations and symbols that will be used throughout this work. A presynaptic neuron is denoted by $i$ and a postsynaptic neuron by $j$. The set of all presynaptic neurons to neuron $j$ is then defined as $\Gamma_j = \{ij| \text{ presynaptic to } j\}$. Similarly, the set of all postsynaptic neurons to the neuron $i$ is defined as $\Gamma_i = \{ji| \text{ postsynaptic to } i\}$.

The moment when a given neuron $j$ emits an action potential will be referred to as the firing
time of that neuron and will be denoted by \( t_j^{(f)} \), with \( f \) varying from 1 to \( n \). The spike train of neuron \( j \) is then characterized by the set of firing times given by equation (2.1).

\[
F_j = \{ s_j^{(1)}, s_j^{(2)},..., s_j^{(n)} \}
\]

where \( t_j^{(n)} \) is the most recent spike of neuron \( j \), also denoted by \( \hat{t}_j \).

The PSP is the response generated by a spike from the presynaptic neuron \( i \) in the postsynaptic neurons \( j \) and has the form shown in figure 2.3. It is important to note that, due to interneural distances and finite axonal transmission times, there may be a delay between the emission of a spike and the beginning of its corresponding PSP. Equation (2.2) gives a mathematical formulation for a generic PSP originated in neuron \( j \) by a spike emitted by neuron \( i \) [GK02b].

\[
e_{ij}(t) = \left[ \exp \left( -\frac{t - t_i^{(f)} - \Delta_{ij}^{ux}}{\tau_m} \right) - \exp \left( -\frac{t - t_i^{(f)} - \Delta_{ij}^{ux}}{\tau_s} \right) \right] H(t - t_i^{(f)} - \Delta_{ij}^{ux}) \]

or in a more simple form [Boh03]

\[
e_{ij}(t) = \frac{t - t_i^{(f)} - \Delta_{ij}^{ux}}{\tau} \exp \left( 1 - \frac{t - t_i^{(f)} - \Delta_{ij}^{ux}}{\tau} \right) H(t - t_i^{(f)} - \Delta_{ij}^{ux})
\]

where \( \tau_m, \tau_s \) and \( \tau \) are time constants, \( \Delta_{ij}^{ux} \) is the axonal transmission delay, and \( H(\cdot) \) is the Heaviside step function. Figure 2.4 is an example of EPSP equation (2.3). Despite both equations have been widely used to simulate PSP, in a more formal and biological approach, the kernel \( e_{ij} \) should be also a function of the last firing time of \( j \), since the PSP depends on the state of the membrane potential, which, in turn, depends on \( \hat{t}_j \). The kernel \( e_{ij}(i - \hat{t}_j, t - t_i^{(f)} - \Delta_{ij}^{ux}) \) can be interpreted as the time course of a postsynaptic potential evoked by the firing of a presynaptic neuron \( i \) at time \( t_i^{(f)} \).

2.2.4 The Spike After Potential – SAP

Immediately after the emission of a spike, the membrane potential does not directly return to the resting potential \( u_{rest} \), but passes through a phase of high hyperpolarization below the resting value [GK02b]. While the action potential is quickly rising or steeply falling, emission of a further spike is impossible. This effect is called absolute refractoriness. The spike after potential (SAP)
Figure 2.4: Example of an EPSP using equation (2.3), with $t_i^{(f)} = 4ms$, $\Delta_{ij}^{ax} = 6ms$ and $\tau = 4ms$.

being negative, makes the emission of a second spike immediately after the first pulse more difficult. This period of reduced sensitivity after a spike is called the relative refractory period. Figure 2.5 shows the membrane potential variation before, during and after the emission of a spike.

2.3 The Equilibrium Potential

The neuron membrane is a nearly perfect electrical insulator and separates the intracellular environment from the surrounding environment. Embedded in the membrane are proteins that act as transporters and ion channels, actively transporting ions from one side to another, resulting in different ion concentrations in each side. This difference in the ion density generates a difference in the electrical potential between both sides of the membrane. At equilibrium, the voltage generated by the different density is given by the Nernst equation [Wil99].

$$E = \frac{kT}{q} \ln \left( \frac{C_{in}}{C_{out}} \right) \tag{2.4}$$

where $E$ is the equilibrium potential (also called Nernst potential), $k$ is the Boltzmann constant, $T$ is the temperature in Kelvin degrees, $q$ is the charge of the ion, and $C_{in}$ and $C_{out}$ are the respective ion concentrations inside and outside the membrane.

For example, the ion concentration of potassium ($K^+$) is higher inside the cell ($\approx 400mM$) than
in the extracellular liquid ($\approx 20mM$). Potassium ions have a positive charge $q = 1.6 \times 10^{-19}$ C. Application of the Nernst equation with the Boltzmann constant $k = 1.4 \times 10^{23}$ J/K yields $E_K = -77mV$ at room temperature. Similarly, the equilibrium potential for sodium ions ($Na^+$) is $E_{Na} = +55mV$ [GK02b]. The equilibrium potential is also called reversal potential because it represents the point where the ions flow reverses its direction to maintain the equilibrium.

In real cells, various ion types are simultaneously present and contribute to the voltage across the membrane. The resting potential $U_{rest}$ is therefore determined by the dynamic equilibrium between the ion flow through the channels (permeability of the membrane) and active ion transport by the ion pumps. When there are more than one permeant ion, equation (2.4) is replaced by equation (2.5), also known as Goldmann equation, which takes into account the permeability of the membrane to each ion of interest [PAF+04, KSJ00].

$$E = \frac{kT}{F} \ln \frac{P_K[K]_2 + P_{Na}[Na]_2 + P_{Cl}[Cl]_1}{P_K[K]_1 + P_{Na}[Na]_1 + P_{Cl}[Cl]_2}$$ (2.5)

In equation (2.5), $P$ indicates the permeability of the membrane, $F$ is the Faraday constant, and the indexes 1 and 2 indicate the ion concentration in- and outside the membrane, respectively. The index of the $Cl^-$ ion is inverted because the $q$ factor of the Nernst equation was eliminated.
The value 58 is for room temperature. Table 2.3 shows the intracellular concentrations of the main ions.

Table 2.1: Extracellular and intracellular ion concentrations for the main ions.

<table>
<thead>
<tr>
<th>Ion</th>
<th>Concentration (mM)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Intracellular</td>
<td>Extracellular</td>
<td></td>
</tr>
<tr>
<td>Squid neuron</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Potassium (K⁺)</td>
<td>400</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Sodium (Na⁺)</td>
<td>50</td>
<td>440</td>
<td></td>
</tr>
<tr>
<td>Chloride (Cl⁻)</td>
<td>40-150</td>
<td>560</td>
<td></td>
</tr>
<tr>
<td>Calcium (Ca²⁺)</td>
<td>0.0001</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Mammalian neuron</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Potassium (K⁺)</td>
<td>140</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Sodium (Na⁺)</td>
<td>5-15</td>
<td>145</td>
<td></td>
</tr>
<tr>
<td>Chloride (Cl⁻)</td>
<td>4-30</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>Calcium (Ca²⁺)</td>
<td>0.0001</td>
<td>1-2</td>
<td></td>
</tr>
</tbody>
</table>

2.4 Neurons Main Characteristics and Features

A last and major step before we can discuss models of spiking neurons is to define the neuro-computational features of real neurons that must be taken into account. In this section we make a brief review of 20 types of real (cortical) neurons response, considering the injection of simple dc pulses [Izh04].

a. **Tonic Spiking**: The neuron fires a spike train as long as the input current is on. This kind of behavior can be observed in the three types of cortical neurons: regular spiking excitatory neurons (RS), low-threshold spiking neurons (LTS) and fast spiking inhibitory neurons (FS).

b. **Phasic Spiking**: The neuron fires only a single spike at the onset of the input.

c. **Tonic Bursting**: The neuron fires periodic bursts of spikes when stimulated. This behavior may be found in chattering neurons in cat neocortex. Since the frequency of bursts may be as high as 50Hz, it is believed that such kind of neuron contributes to the gamma-frequency oscillations in the brain.

d. **Phasic Bursting**: The neuron fires only a single burst at the onset of the input.

e. **Mixed Model**: The neuron fires a phasic burst at the onset of stimulation and then switch to the tonic spiking mode. The intrinsically bursting excitatory neurons in mammalian neocortex may exhibit this behavior.
f. **Spike Frequency Adaptation:** The neuron fires tonic spikes with decreasing frequency. RS neurons usually exhibit adaptation of the interspike intervals, when these intervals increase until a steady state of periodic firing is reached, while FS neurons show no adaptation [GK02b].

g. **Class 1 Excitability:** Here, the frequency of tonic spiking depends on the strength of the input, i.e., the weaker is the dc current, the lower will be the spike frequency. The frequency of tonic spiking of neocortical neurons may vary from 2 to 200MHz.

h. **Class 2 Excitability:** The neuron is unable to fire low-frequency spike trains and will either fire a high frequency spike train or will not fire at all.

i. **Spike Latency:** The neuron fires with a delay that depends on the strength of the stimulation. RS neurons in mammalian cortex can have latencies of tens of milliseconds.

j. **Subthreshold Oscillations:** The neuron exhibits oscillatory potentials.

k. **Frequency Preference and Resonance:** Due to the resonance phenomenon, a neuron having oscillatory potentials can respond selectively to the inputs having frequency content similar to the frequency of subthreshold oscillations. Such neurons are called *resonators* [Izh01].

l. **Integration and Coincidence Detection:** The neuron without oscillatory potentials acts as an integrator [Izh01]. The higher the frequency, the more likely it fires.

m. **Rebound Spike:** The neuron fires a post-inhibitory (rebound) spike when it receives and then is released from an inhibitory input.

n. **Rebound Burst:** The neuron fires post-inhibitory bursts. It is believed that such bursts contribute to the sleep oscillations in the thalamo-cortical system.

o. **Threshold Variability:** A biological neuron has a variable threshold that depends on the prior activity of the neurons. Therefore, it may not fire in response to a single brief excitatory pulse, but will fire if a brief inhibitory input is applied just before the same excitatory pulse.

p. **Bi-stability of Resting and Spiking States:** The neuron exhibits two stable modes of operation: resting and tonic spiking (or even bursting). The modes are switched by an excitatory or inhibitory pulse.

q. **Depolarizing After-Potentials:** After firing a spike, the membrane potential of a neuron may exhibit a prolonged after-hyperpolarization (AHP) or a prolonged depolarized after-potential (DAP). In any case, such a neuron has shortened refractory period and becomes superexcitable.
Figure 2.6: Examples of all neuro-computational features discussed in this section. The input current is plotted beneath the neuronal response and the horizontal bar denotes 20ms. No model may exhibit all these 20 neurocomputational properties simultaneously as some of them are mutually exclusive. All the neuronal responses showed here were obtained using a simple spiking model with only four easily tunable parameters [Izh03]. Electronic version of this figure and reproduction permissions are freely available at www.izhikevich.com.
r. **Accommodation**: The neuron loses its sensitivity to a strong but slowly increasing stimulus (flat ramp) and does not fire, although it may fire in response to a weak but fast increasing stimulus (step ramp).

s. **Inhibition-Induced Spiking**: The neuron remains quiescent as long as there is no input, but fires when hyperpolarized by an inhibitory input or an injected current.

t. **Inhibition-Induced Bursting**: Similar to the previous case, the neuron can fire tonic bursts of spikes in response to a steady hyperpolarization.

### 2.5 Conclusions

After this brief presentation we can evaluate what to simulate in order to have a spiking neuron model. It is also possible to have an idea about how complex a nervous system can be, even that of a simple animal. Now, once we know what characteristics to simulate, the next chapter will introduce a few mathematical neuronal models, which will permit us to simulate these characteristics with varying degrees of details, ranging from the simple *integrate-and-fire* model to the far more complex *compartmental model*. 


Chapter 3

Spiking Neuron Models

3.1 Introduction

Since neural activity may be described at several levels of abstraction, the choice of the model depends on this level. On the one hand, considering the microscopic level, there is a large number of ion channels, pores in the cell membrane which open and close depending on the voltage and the presence (or absence) of various chemical messenger molecules. Compartmental models, where each small segment of a neuron is described by a set of ionic equations, aim at a description of these processes. On the other hand, when choosing a higher level of abstraction, there is no need to worry about the spatial structure of a neuron or about the exact ionic mechanisms. We only have to consider the neuron as a simple homogeneous unit, which generates spikes if the total excitation is sufficiently large, as with rate models. It is also clear that the choice of which model to use is a very important decision that must be based on what kind of phenomena or neuronal behavior we wish to simulate.

In this rather superficial explanation, we divided the spiking neuron models into three main classes, namely threshold-fire, conductance based and compartmental models [GK02b]. Each one of these classes will be treated here, with emphasis on the first two, as the third is quite complex to fit in the scope of this work. A fourth class, namely rate model [Maa97, Ger99], was also considered, although it may not be classified as a genuine spiking model.

3.2 Threshold-Fire Models

The threshold-fire models are based on the temporal summation of all contributions to the membrane potential $u(t)$ received from all presynaptic neurons. If this contribution exceeds a threshold $\theta$, then the postsynaptic neuron will fire. In the simplest case one assumes that the PSPs simply sum up linearly in the soma. In this section we will discuss two of these models, the integrate-
and-fire and the spike response model – SRM [Ger01, GK02b].

3.2.1 Integrate-and-Fire Model – I&F

The integrate-and-fire neuron [MB98] is perhaps the most used and well-known example of a formal spiking neuron model. The basic model is also called leaky-integrate-and-fire (LIF) [GK02b] because the membrane is assumed to be leaky due to ion channels, such that after a PSP the membrane potential approaches again a reset potential \( u_{rest} \). This model is also known as the linear integrate-and-fire neuron.

![Figure 3.1: Integrate-and-fire neuron (from [MB98]).](image)

The model is composed by the RC circuit shown inside the circle on the right-hand side of the diagram in figure 3.1. The circuit is charged by an input current \( I \). If the voltage across the capacitor \( C \) reaches a threshold \( \vartheta \), the circuit is shunted and a pulse is transmitted to other neurons (lower right). A pulse sent out by a presynaptic neuron and travelling on the presynaptic axon (left), is low-pass filtered first (middle) before it is fed as a current pulse \( \delta(t-t_i^{(f)}) \) into the integrate-and-fire circuit. The voltage response of the RC circuit to the presynaptic pulse is the postsynaptic potential \( \epsilon(t-t_i^{(f)}) \). The driving current splits into two components, one charging the capacitor, the other going through the resistor, yielding the equation (3.1).

\[
I(t) = \frac{u(t)}{R} + C \frac{du}{dt}
\]  

(3.1)

Making the time constant \( \tau_m = RC \) we can rewrite 3.1 in the form of equation (3.2),

\[
\tau_m \frac{du}{dt} = -u(t) + RI(t)
\]  

(3.2)
where $\tau_m$ is the membrane time constant. It is easy to see that the voltage $u(t)$ across the capacitor $C$ is the membrane potential.

Since equation (3.2) is a first-order linear differential equation and cannot describe full neuronal spiking behavior, it must be supplemented by a threshold condition. A pulse will be emitted at $t = t^{(f)}$ when the threshold $\theta$ is reached, i.e., when $u(t^{(f)}) = \theta$. The form of the spike is not described explicitly, only the firing time $t^{(f)}$ is used. Immediately after $t^{(f)}$, the potential is reset to $u_{\text{rest}}$ and the integration starts over again from this point [MB98, GK02b]. The reset potential is defined by equation (3.3).

$$\lim_{t \to t^{(f)} ; t > t^{(f)}} u(t) = u_{\text{rest}} \quad (3.3)$$

The model behavior is then ruled by equations (3.2) and (3.3).

![Graphs](image)

Figure 3.2: **Left:** Example of the membrane potential of an integrate-and-fire neuron excited by a constant input current $I_0 = 2mA$, with $\tau = 10ms$, $u_{\text{rest}} = 0$ and $R = 1\omega$. **Right:** Variation of the firing rate as a function of the exciting current in an integrate-and-fire neuron, with the same parameters. In the solid line curve no absolute refractoriness was used and in the dashed line curve a $4ms$ absolute refractoriness was used.

Considering now an integrate-and-fire neuron with a constant input current $I_0$ and a rest potential $u_{\text{rest}} = 0$ and integrating (3.2) with the initial condition $u(t^{(1)}) = u_{\text{rest}} = 0$, we obtain equation (3.4).

$$u(t) = R I_0 \left[ 1 - e^{-\frac{t}{\tau_m}} \right] \quad (3.4)$$
Assuming that a spike occurred at $t = t^{(1)}$, no further spike will occur as long as $u(t) < \theta$. For the production of a spike at $t = t^{(2)}$ we must have the condition $u(t^{(2)}) = \theta$, or

$$\theta = R I_0 \left[ 1 - e^{-\frac{R I_0}{\theta}} \right]$$  \hspace{1cm} (3.5)

Making $T = t^{(2)} - t^{(1)}$ and solving (3.5)

$$T = \tau_m \ln \frac{R I_0}{R I_0 - \theta}$$  \hspace{1cm} (3.6)

We can see that the neuron will keep firing with period $T$ as long as the constant current $I_0$ is applied. To take into account an absolute refractoriness $\Delta^{abs}$, we just have to add it to the period $T$. Since in this case the frequency is the mean firing rate $\nu$, we have

$$\nu = \left[ \Delta^{abs} + \tau_m \ln \frac{R I_0}{R I_0 - \theta} \right]^{-1}$$  \hspace{1cm} (3.7)

![Graph](image)

Figure 3.3: Example of the membrane potential of a linear integrate-and-fire neuron excited by a variable input current, with $\tau = 10 ms$, $u_{rest} = 0$ and $R = 1 \omega$. As suggested in [GK02b], the input current is the sum of four sinusoidal components with randomly chosen frequencies and a positive current $I_0 = 1.5 mA$.

A variation of the linear integrate-and-fire model includes a second linear equation, describing the dynamics of a high-threshold $K^+$ ionic current, enabling the spike frequency adaptation. Other
variations are the integrate-and-fire-or-burst neuron [Izh04, Coo03, Rul02], used to model cortical neurons, and the resonate-and-fire model [Izh01].

![Graph of membrane potential](image)

Figure 3.4: Example of the membrane potential of a quadratic integrate-and-fire neuron excited by the same input showed in figure 3.3. Here $u_{rest} = 0$ and $u_c = 0.5$.

A more general form of equation (3.1), known as the nonlinear integrate-and-fire neuron, is given by equation (3.8),

$$\tau \frac{du}{dt} = F(u) + G(u)I$$

(3.8)

where $F(u)$ is a voltage dependent decay constant and $G(u)$ is a voltage dependent input resistance. A variation of this model is the quadratic integrate-and-fire model [GK02b, Izh04], described by equation (3.9).

$$\tau \frac{du}{dt} = a_0(u - u_{rest})(u - u_c) + RI$$

(3.9)

where $a_0 > 0$ is a constant and $u_c > u_{rest}$ is a critical voltage, above which $\frac{du}{dt} > 0$. This model is also known as Θ-neuron or the Ermentrout-Kopell canonical model (when written in its trigonometrical form) [Izh04]. This model has spike latencies, activity-dependent threshold and bi-stability of resting and tonic spiking modes [Izh04].
3.2.2 Spike Response Model – SRM

The state of a neuron $j$ is described by a single state variable $u_j(t)$. If $u_j(t)$ crosses from below ($\frac{du_j}{dt} > 0$) a threshold $\theta$ at a moment $t^{(f)}_j$, then a spike is generated. As already stated in equation (2.1), the set of all firing times of neuron $j$ is denoted by

$$F_j = \{ t^{(f)}_j ; 1 \leq f \leq n \} = \{ t | u_j(t) = \theta \}$$ (3.10)

We can see in figure 2.5 that there are two different processes influencing the value of the state variable $u_j(t)$. The first is the contribution of all presynaptic neurons $i \in \Gamma_j$, which models the neuron response to presynaptic spikes. It is given by equation (3.11).

$$h(t) = \sum_{i \in \Gamma_j} \sum_{t^{(f)}_i \in \Gamma_j} w_{ij} e_{ij} (t - \hat{t}_j, t - t^{(f)}_i - \Delta_{ij}^{ax}) + I_{ij}^{ext}$$ (3.11)

where $w_{ij}$ is the synaptic efficacy, $I_{ij}^{ext}$ is an external excitation, and $e_{ij}$ is the PSP, as shown in figure 2.4 and equations (2.2) and (2.3). The postsynaptic potential can either be excitatory (EPSP) or inhibitory (IPSP). In most cases, $\Delta_{ij}^{ax}$ is neglected and does not appear in the equations. To avoid confusion, we will use $s = t - t^{(f)}_j - \Delta_{ij}^{ax}$.

The second process is the reset of $u_j(t)$ till it reaches $u_{\text{rest}}$, including the negative overshoot which typically follows a spike (the refractory period). This term describes the response of neuron $j$ to its own spikes and is modeled by a response kernel $\eta_j(t - \hat{t}_j)$, which is a function only of the last firing time of $j (\hat{t}_j)$. More formally, all previous firing times of $j$ should be considered, as in $\sum \eta_j(t - t^{(f)}_j)$, but the approach adopted above does not alter the results [MB98].

Now, if we take into account the linear response of the membrane potential to an external driving current $I^{ext}$, the expression for the last term of (3.11) is then given by equation (3.12).

$$\int_0^\infty \kappa(t - \hat{t}_j, s) I^{ext}(t - s) ds$$ (3.12)

The total postsynaptic potential is also

$$h(t|\hat{t}_j) = \sum_i w_{ij} \sum_{t^{(f)}_i} e_{ij}(t - \hat{t}_j, s) + \int_0^\infty \kappa(t - \hat{t}_j, s) I^{ext}(t - s) ds$$ (3.13)

This dependence of the elapsed time since the last spike at $\hat{t}_j$ can be explained by the fact that immediately after a spike, many ion channels are open so that the membrane resistance is reduced.
Figure 3.5: This is the same as figure 2.5, but the threshold was altered to show the effect of the dynamic threshold.

The state $u_j(t)$ of model neuron $j$ at time $t$ is given by the linear superposition of all contributions and can be written as in equation (3.14).

$$u_j(t) = \eta(t - \hat{t}_j) + h(t; \hat{t}_j)$$  \hspace{1cm} (3.14)

Besides the use of the $\eta$ kernel, refractoriness can be also modeled by an increase in the threshold after a spike so that $\vartheta \rightarrow \vartheta(t - \hat{t}_j)$. If the objective is preventing the neuron from firing, an increase in the threshold is equivalent to the hyperpolarization of the membrane potential, as can be seen in figure 3.5. This is called dynamic threshold [MB98, GK02b].

A simplified Spike Response Model, SRM$_0$ for short [MB98], can be constructed with little modifications to the complete model, as shown in figure 3.6. First we suppose that $\epsilon_j$ is the same for all presynaptic neurons and independent of the first argument $t - \hat{t}_j$. Then we make the $\eta$ kernel also independent of the first argument, so we can rewrite (3.14) as equation (3.15).

$$u_j(t) = \eta(t - \hat{t}_j) + \sum_i w_{ij} \sum_{q \neq j} \epsilon_0(s) + \int_0^\infty \kappa_0(s) I_{ext}(t-s) ds$$  \hspace{1cm} (3.15)
Figure 3.6: The SRM₀ model, where the spike was replaced by a δ-pulse and Δₓ = 0 (from [GK02b]).

It can be demonstrated that the Spike Response Model, with the appropriate kernels, is equivalent to the integrate-and-fire model. Furthermore, with a different choice of kernels, the Spike Response Model also approximates the Hodgkin-Huxley equations with time-dependent input [MB98, GK02b].

3.3 Conductance-Based Models

This is a quite complex class of neuron models, which are usually based on the simulation of the intricate behavior of ionic channels. As these channels open and close, their conductances change accordingly, yielding a set of differential equations describing the process. The variations among the models in this class is mostly due to the choice of which channels to use and the parameters of the resulting differential equations. This section will be focused on the classic model of Hodgkin and Huxley [HH52] and some of its variations.

3.3.1 Hodgkin-Huxley Model

The Hodgkin-Huxley model is generally accepted as the classic description of a conductance model. Hodgkin and Huxley [HH52], based on their experiments with the giant axon of the squid, found three types of ionic current, namely sodium (Na), potassium (K) and a leak current. The first two types are controlled by specific voltage dependent ion channels. The third type takes
care of other channels and consists mainly of $\text{Cl}^-$ ions.

Taking figure 3.7 as a reference, the model can be described as a RC-circuit. The cell membrane is a good insulator and acts as a capacitor. Besides the capacitor, there are three resistances, one for each ion channel considered in the model. The potassium and sodium channels are voltage dependent. As explained before in section 2.3, the different ion concentrations inside and outside the cell yields the Nernst Potential, represented by the batteries. As the total current must be equal to the sum of the currents of the branches, we have
\[ I(t) = I_c(t) + \sum_{ch} I_{ch}(t) \]  

(3.16)

Since \( I_c = C \frac{du}{dt} \)

\[ C \frac{du}{dt} = -\sum_{ch} I_{ch} + I(t) \]  

(3.17)

The membrane potential is given by \( u(t) \) and \( \sum_{ch} I_{ch} \) is the sum of the ionic currents passing through the cell membrane.

Table 3.1: Parameters of the Hodgkin-Huxley equation. The usual units used for these parameters and variables are \( mS/cm^2 \) for conductances, \( mV \) for the equilibrium potentials, and \( \mu F \) for capacitances.

<table>
<thead>
<tr>
<th>( x )</th>
<th>( E_x(mV) )</th>
<th>( g_x(mS/cm^2) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Na</td>
<td>50</td>
<td>120</td>
</tr>
<tr>
<td>K</td>
<td>-77</td>
<td>36</td>
</tr>
<tr>
<td>L</td>
<td>-54.4</td>
<td>0.3</td>
</tr>
</tbody>
</table>

Table 3.2: Parameters of the equations for \( \alpha_x \) and \( \beta_x \).

<table>
<thead>
<tr>
<th>( x )</th>
<th>( \alpha_x(u/mV) )</th>
<th>( \beta_x(u/mV) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m )</td>
<td>( (2.5 - 0.1u)/[\exp(2.5 - 0.1u) - 1] )</td>
<td>( 4\exp(-u/18) )</td>
</tr>
<tr>
<td>( h )</td>
<td>( 0.07\exp(-u/20) )</td>
<td>( [1/\exp(3 - 0.1u) + 1] )</td>
</tr>
<tr>
<td>( n )</td>
<td>( (0.1 - 0.01u)/[\exp(1 - 0.1u) - 1] )</td>
<td>( 0.125\exp(-u/80) )</td>
</tr>
</tbody>
</table>

Characterizing each channel by its conductance (or resistance), the leak conductance \( (g_L) \) is constant and the other two \( (g_K \) and \( g_{Na} \) are time and voltage dependent. Furthermore, considering that a channel is either open or closed, three more variables were added [HH52], yielding equation (3.18) for the sum of the three currents.

\[
\sum_{ch} I_{ch} = g_{Na} m^3 h (u - E_{Na}) + g_K n^4 (u - E_K) + g_L (u - E_L) \]  

(3.18)

If the channels were always open, the currents could be defined simply by \( I_x = g_x u \) [Koc99]. Equation (3.17) can then be written as
\[
C \frac{du}{dt} = -[g_N e m^3 h (u - E_N) + g_K n^4 (u - E_K) + g_L (u - E_L)] + I(t) \tag{3.19}
\]

The three new variables are given by the following differential equations

\[
\frac{dm}{dt} = a_m(u)(1 - m) - \beta_m(u)m
\]

\[
\frac{dn}{dt} = a_n(u)(1 - n) - \beta_n(u)n
\]

\[
\frac{dh}{dt} = a_h(u)(1 - h) - \beta_h(u)h
\]

The functions \( a \) and \( \beta \), shown in table 3.3.1, are empirical equations originated from the adaptations made to fit the experimental data. Equations (3.19) and (3.20) define the Hodgkin-Huxley model. Figure 3.9 shows the equilibrium functions and the time constants for the variables \( m, n, \) and \( h \). The values for \( m_0, n_0, h_0, \tau_m, \tau_n, \) and \( \tau_h \) are given by equations (3.20).

\[
x_0 = \frac{a_x(u)}{[a_x(u) + \beta_x(u)]} \tag{3.20}
\]

\[
\tau_x = \frac{1}{[a_x(u) + \beta_x(u)]}
\]

where \( x \) stands for \( m, n \) or \( h \). Equation (3.20) is usually also written as

\[
\frac{dx}{dt} = \frac{x_0 - x}{\tau_x} \tag{3.21}
\]

With the previous equations and the right set of parameters, Hodgkin and Huxley were able to describe the enormous amount of data collected from experiments on the giant axon of the squid [Has00]. After this successful first step, there have subsequently been several modifications to the original equations in order to describe other experimental situations, giving origin to other models. These models differ mainly in the additional ion channels that have to be included (or excluded), specially those that account for \( Ca^{2+} \) and the slow components of the potassium current. As a general rule, for each type of ion channel \( i \), a current \( I_i = g_i x_i^n (u - E_i) \) is added.
Figure 3.9: **Left**: Steady-state activation (m for sodium and n for potassium) and inactivation (h for sodium). **Right**: time constants as a function of the membrane potential. (from [Koc99, GK02b]).

[MB98, Wil99, Koc99]. Here \( x \) is a variable with dynamics, like \( m, n, \) or \( h; \) \( g_i \) is the conductance parameter; \( n_i \) is an appropriate exponent; and \( E_i \) is the correspondent equilibrium potential.

Figure 3.10: Example of the FitzHugh-Nagumo model with parameters \( \alpha = 0.3, \epsilon = 0.03, \gamma = 0.4, \) with an excitation current of 0.5\( \mu A \) and a threshold of 0.8mV.

### 3.3.2 FitzHugh-Nagumo Model

The FitzHugh-Nagumo model [Fit61, NAY62], described by equations (3.22), is a simplification of the Hodgkin-Huxley model where the set of four equations is reduced to a system of two equations, i.e., a cubic equation representing the fast variable \( (u) \) and a linear equation for the slow
variable \(\psi\).

\[
e \frac{du}{dt} = -u(u - \alpha)(u - 1) - \psi + I
\]

(3.22)

\[
\frac{d\psi}{dt} = u - \gamma \psi
\]

The advantage of the reduction of the number of equations to a two-dimensional set of equations is that it allows a systematic phase plane analysis of the model. The two-dimensional set of equations is also called a Bonhoeffer/Van der Pol oscillator [Wil99].

3.3.3 Morris-Lecar Model

This 2D model was suggested by Morris and Lecar [ML81] to describe oscillations in a barnacle giant muscle fiber. The model is given by equations (3.23), where one expression is for the membrane potential \(u\) and the other for the potassium gating variable \(n\):

\[
C \frac{du}{dt} = I - g_{Ca} m_{\infty}(u)(u - E_{Ca}) - g_{Kn}(u - E_{K}) - g_{L}(u - E_{L})
\]

\[
\frac{dn}{dt} = \frac{\phi [n_{\infty}(u) - n]}{\tau_n(u)}
\]

(3.23)

where the steady states for Ca\(^{2+}\) and K\(^+\) and the time constant \(\tau_n\) are:

\[
m_{\infty} = \frac{1}{1 + \exp[-(u - u_1)/u_2]}
\]

(3.24)

\[
n_{\infty} = \frac{1}{1 + \exp[-(u - u_3)/u_4]}
\]

(3.25)

\[
\tau_n(u) = \frac{1}{\cosh \left( \frac{u - u_2}{2u_4} \right)}
\]

(3.26)

Sometimes \(\tau_n\) is referred to as \([\lambda_n(u)]^{-1}\). Except for a change in the parameters, \(m_{\infty}\) and \(n_{\infty}\) are functionally identical.
3.3.4 Hindmarsh-Rose Model

The general form of the Hindmarsh-Rose model of thalamic neuron [HR82, HR84] is given by equations (3.27).

\[
\frac{du}{dt} = v - F(u) + I - w
\]
\[
\frac{dv}{dt} = G(u) - v
\]
\[
\frac{dw}{dt} = \frac{[H(u) - w]}{\tau}
\]

(3.27)

where $F$, $G$ and $H$ are specific functions that must be correctly chosen to produce the desired results. Depending on their choice, the model can exhibit all the properties shown in figure 2.6 [Izh04]. For instance, equations (3.28) were used in [BC84].

\[
\frac{du}{dt} = v - au^3 + bu^2 + I - w
\]
\[
\frac{dv}{dt} = c - du^2 - v
\]
\[
\frac{dw}{dt} = \frac{s(u - u_0) - w}{\tau}
\]

(3.28)

where $F(u) = au^3 - bu^2$, $G(u) = c - du^2$, and $H(u) = s(u - u_0)$.

3.3.5 Izhikevich Model

This model, proposed in [Izh03], can exhibit all the properties shown in figure 2.6. The model uses equations (3.29).

\[
\frac{du}{dt} = 0.04u^2 + 5u + 140 - w + I
\]
\[
\frac{dw}{dt} = a(bu - w)
\]

(3.29)
Figure 3.11: Example of the Izhikevit model with parameters $a = 0.02$, $b = 0.2$, $c = -65$, and $d = 8$, with an excitation current of $50\mu A$ and a threshold of $30\text{mV}$.

with the resetting after-spike

$$\text{if } u \geq +30\text{mV, then } \begin{cases} u \leftarrow c \\ w \leftarrow w + d \end{cases}$$ (3.30)

The variable $u$ represents the membrane potential and $w$ is a membrane recovery variable, accounting for the activation of $K^+$ ionic currents and inactivation of $Na^+$ ionic currents. The model can exhibit firing patterns of all known types of cortical neurons with the choice of parameters $a$, $b$, $c$, and $d$ given in [Izh03].

### 3.4 Compartmental Models

Since Ramón y Cajal [yC03] first described a neuron, the known complexity of this biological structure has been steadily increasing. This complexity is better captured by the compartmental models, which take into account the spatial structure of the dendritic tree and also model the synaptic transmission at a greater level of detail. With this approach, it is possible to consider
other ion currents, beyond the \(Na^+\) and \(K^+\) currents incorporated in the Hodgkin-Huxley model (\(Ca^{2+}\) and Ca-mediated K currents which are related to neuronal adaptation [MB98]). Entire books have been written on this subject and it is not our aim to present a complete description of these neuron models, but just to introduce its basic concepts.

![Electrical Circuit Diagram](image)

**Figure 3.12:** A "generic" equivalent circuit of a neural compartment (from [BB03]).

As already mentioned previously, the dendrites receive synaptic inputs from other neurons. These inputs activate ion channels in the dendrites, which create postsynaptic potentials that are propagated to the cell body (soma), where voltage-activated ion channels may create action potentials. Usually these channels are located in a region called axon hillock, at the base of the soma [KSJ00, PAF+04]. The axon then propagates the action potentials to its terminal branches, where they form synapses connecting to other neurons.

The basic idea of compartmental models is to divide these components into smaller uniform components or *compartments*. Each compartment is then modeled with equations describing the equivalent electrical circuit. The use of appropriate differential equations for each compartment enables the simulation of their behavior as well as their interactions with other compartments. This notion of an equivalent electrical circuit for a small piece of cellular membrane is the basis for all compartmental models [BB03].

An equivalent circuit is shown in figure 3.12, where \(u\) is the membrane potential, which appears across the membrane capacitance \(C\). It may cause a current flow through the axial resistance \(R_a\) (or \(R'_a\)) into or out the compartment when there is a difference \(u - u''\) (or \(u - u'\)) between the two compartments. The resistor \(G_{ch}\) crossed by an arrow represents a generic variable channel conductance, specific to a particular ion combination. The corresponding equilibrium potential is represented by the battery in series with the conductance. The other resistor is the *leakage con-

33
ductance $G$, and the associated equilibrium potential $E$ is typically close to $u_{\text{rest}}$. The current $I$ represents an external input. The resulting differential equation is (3.31).

$$C \frac{du}{dt} = (E - u)G + \sum_{ch}(E_{ch} - u)G_{ch} + \frac{(u' - u)}{R_a} + \frac{(u'' - u)}{R_a} + I$$  \hspace{1cm} (3.31)

Equation (3.31) describes a linear compartmental model. Other more complex compartmental models would also contain non-linear conductances for various ionic currents, or could model synaptic currents at a greater level of detail.

3.5 Rate Models

The rate models, also known as sigmoidal units, are the most traditional and widely used models for the analysis of learning and memory in artificial neural networks.

As already mentioned, the first choice to be made before the construction of a neuronal model is the level of abstraction and, therefore, the complexity of the model. The class of rate models, first proposed in [MP43], represents the highest abstraction level, as they neglect the pulse structure of the neuronal systems and the neural activity is described only in terms of spike rate. A good example of rate model is the Perceptron [Ros58, MP88].

The response $v_j$ of a neuron $j$ is also a rate and depends on a sigmoidal function. The input to this function is the summation of all the excitations applied to $j$, i.e., all the output rates of the presynaptic neurons $i \in \Gamma_j$:

$$v_j = g \left( \sum_{i \in \Gamma_j} w_{ij} v_i \right)$$  \hspace{1cm} (3.32)

where $w_{ij}$ is the synaptic strength between neurons $i$ and $j$. Equation (3.32) is the starting point of standard neural network theory [Hay99, BLdC00].

The dependence of a sigmoidal function may be explained by the equation (3.33), known as Naka-Rushton function [NR66], which relates the stimulus intensity $p$, the net postsynaptic potential, to the response or spike rate of a neuron observed in several visual areas (lateral geniculate, striate cortex, middle temporal cortex).

$$v(p) = \begin{cases} 
\frac{M p^n}{\sigma^n + p^n} & \text{for } p \geq 0 \\
0 & \text{for } p < 0 
\end{cases}$$  \hspace{1cm} (3.33)
In equation (3.33), \( v \) is the spike rate response to the stimulus \( p \); \( M \) is the maximum spike rate for very intense stimuli; \( \sigma \) determines the point where \( v(p) \) reaches half its maximum (semi-saturation point); and \( n \) determines the maximum slope of the function. A plot of this function is shown in figure 3.13, for \( n = 2, 4 \) and 6, \( M = 100 \), and \( \sigma = 50 \).

![Graph of equation (3.33) for \( M = 100 \), \( \sigma = 50 \), and \( n = 2, 4 \) and 6.]

Figure 3.13: A plot of equation (3.33) for \( M = 100 \), \( \sigma = 50 \), and \( n = 2, 4 \) and 6.

### 3.6 Conclusions

As one can grasp from the previous material, a real neuron is a very complex entity and its mathematical modelling may become a quite challenging task, depending on how deep into the details we intend to go through. This depth is determined by the behaviors and features the modeled neurons must have. The models mentioned here are but a few among several other models used for general or special purposes, e.g., [IMI99, CBC02, RH05, SKA01, SS02, TNS02].

In the following chapters the models presented here will be explored and some kind of neural dynamics will be studied.
Chapter 4

Neurodynamics

4.1 Introduction

In the previous chapters we reviewed the basic components of the nervous systems, giving a
glimpse of all their underlying complexity. Following, we then discussed some of the mathemat-
ical models that permit us to simulate these systems, from the more simple to the complex ones.
The objective of this chapter is to briefly discuss those aspects tied to the practical application of
the neuron models to simulate the dynamical behavior of real neural systems [And04].

The complete dynamical behavior of a neural system is usually described by high-dimensional
nonlinear differential equations, whose results are difficult to visualize, and even more difficult
to analyze [Kin91]. Fortunately, there are some models that are a reduction of the more complex
models to two dimensions, like the Morris-Lecar [ML81] or the FitzHugh-Nagumo [Fit61, NAY62]
models. The use of these simplified equations makes it easier (or less difficult) to study and sim-
ulate the nonlinear phenomena of neural systems, like bifurcation and hysteresis.

The nature of the stability and steady states of a nonlinear system can be determined by the
eigenvalues of the associated Jacobian matrix, evaluated at the equilibrium points [ASY00]. Given
a nonlinear system described by the equation

\[
\frac{dX}{dt} = F(X)
\]

then the equilibrium point \( X_{eq} \) is a solution to \( F(X_{eq}) = 0 \) [ASY00].

One of the simplest equations that have been proposed for spike generation are the FitzHug-
Nagumo equations (3.22), presented here with explicit time constants as equations (4.1). It is worth
of noting that the time constant for \( u \) is usually more than ten times faster than that for \( w \), reflecting
the fact that the process in the axon is faster than the recovery process [Wil99].

36
\[
\frac{du}{dt} = \frac{1}{\tau_u} \left( u - \frac{u^3}{K_1} - w + I \right) \\
\frac{dw}{dt} = \frac{1}{\tau_w} (-w + K_2u + K_3)
\] (4.1)

Figure 4.1: Phase plane showing the limit cycle (bold line) and nullclines for \( \frac{du}{dt} = 0 \) (solid line) and \( \frac{dw}{dt} = 0 \) (solid line) using the FitzHugh-Nagumo equations, with an external stimulus of \( I = 1.5 \) and parameters \( \tau_u = 0.09 \), \( \tau_w = 1.20 \), \( K_1 = 3 \), \( K_2 = 1.25 \) and \( K_3 = 1.50 \).

To give a geometric picture of the dynamics of a model, the use of phase plane (or phase space) is quite helpfull. A phase plane is the plot of pairs of points \((u(t), w(t))\) for a range of \( t \) values, as shown in figure 4.1 for the FitzHug-Nagumo equations, with an external input \( I = 1.5 \). In this figure we can see the nullclines, drawn for the values of \( u \) and \( w \) for which the right sides of equations (4.1) are zero, and an equilibrium point, where the nullclines cross each other. In this case there is only one equilibrium point. Also shown in figure 4.1 is a limit cycle, which is a closed orbit in the phase plane, indicating periodic behavior [DA01].

4.2 Example 1: Hysteresis

The hysteresis is an important nonlinear phenomenon, often seen in the presence of a subcritical Hopf bifurcation [ASY00, HI01]. Concerning neural systems, its existence was practically demonstrate in a squid axon [GLR80] by stimulating it with a current that began at 0 and increased linearly up to a maximum value.
Figure 4.2: Hysteresis in the squid axon produced in response of a triangular current ramp, with values ranging from \( I = 0 \) to \( I = 0.25mA \) (showed here out of scale, only for reference). The parameters used in equations (4.2) were \( C = 0.8\mu F/cm^2 \) and \( \tau = 1.9 \)

\[
C \frac{du}{dt} = -(17.81 + 47.7u + 32.63u^2)(u - 0.55) - 26.0w(u + 0.92) + I
\]

\[
dw \quad \quad = \quad \quad \frac{1}{\tau}(-w + 1.35u + 1.03)
\]

(4.2)

To emphasize the importance of neural modeling, we repeat this experiment using equations (4.2) [Wil99]. An illustration of the results is shown in figure 4.2. Hysteresis occur because the present state of a nonlinear system is determined not only by the present state of the stimulation but also by the history of stimulation.

4.3 Example 2: Class 1 and Class 2 Excitabilities

One of the features of a neuron described in chapter 2 was class 1 and class 2 excitabilities. Almost all mammalians neurons are of the first type [KSJ00], while the squid axon is of the second class. Neurons of the class 1 may fire at arbitrary low rate and those of class 2 do not. The ability to fire at low rates is attributed to a rapid transient \( K^+ \) current [Wil99], which is found in a wide range of neurons, including mammalian and human neocortical neurons. Using equations (4.2) slightly modified to accommodate a quadratic equation for the recovery variable \( w \), yielding equations
Figure 4.3: Spike train with a rate of 4.9 spikes per second, showing a class 1 excitable neuron (4.3), it is possible to simulate a slow firing neuron. These new equations permit the approximation of many effects of the transient current [RH89]. The results obtained with the parameters values used here approximate those exhibit by a human neocortical neuron [Wi99].

Figure 4.3 shows a spike train with a rate of 4.9 spikes per second produced by equations (4.3) with a capacitance \( C = 1.0 \mu F/cm^2 \), an input current \( I = 0.22 nA \) and a time constant \( \tau = 5.6 ms \).

\[
\frac{C}{d} \frac{du}{dt} = -(17.81 + 47.58u + 33.8u^2)(u - 0.48) - 26.0w(u + 0.95) + I
\]

\[
\frac{d}{dt} \frac{dw}{dt} = \frac{1}{\tau} (-w + 1.29u + 0.79 + 0.33(u + 0.38)^2)
\]

(4.3)

Using again equations (4.3) with the same parameters values as above and applying an input current \( I = 0 \) and \( I = 0.6 nA \), it is possible to illustrate the occurrence of a saddle-node bifurcation. The results for both current values are plotted in figure 4.4, where we can see three steady states for \( I = 0 \) (when the nullclines cross each other), namely a stable node, a saddle point, and an unstable spiral point.

As \( I \) is increased the behavior of the systems changes qualitatively and the nullclines are modified so that the asymptotically stable node and the saddle point move toward each other and, at a specific transition point, both coalesce and vanish and only the unstable spiral point remains [GK02b] (see figure 4.4). This transition point between stationary and oscillatory states is called bifurcation and, in this special case, a saddle-node bifurcation. Because of the dependence of a bifurcation on the value of \( I \), it is some times called the bifurcation parameter.
Figure 4.4: Phase plane showing the limit cycle and nullclines for the human neuron for $I = 0$ (left) and $I = 0.6nA$ (right).

Comparing figures 4.1 and 4.4 it is easy to see that in the first case there is only one fixed point, while in the second there are three. In equations 4.1) the only fixed point changes from stable to unstable at a critical value of $I$, denoting the occurrence of a (subcritical) Hopf bifurcation [GK02b]. The difference between these two situations is that with only one fixed point the transition to the oscillatory state occurs with a nonzero frequency (class 2), and thus with a discontinuity in the gain function (spike rate versus $I$). In the second case, the transition is smooth (class 1) and the gain function is continuous [HI01] (see figure 4.5).

Figure 4.5: Gain function for Class 1 and 2 neurons. The class 1 curve is a continuous function, while the class 2 curve has a discontinuity.
4.4 Example 3: Neural Chaos

Applying a periodic stimulus \( i(t) = I_0 + A \sin(2\pi \omega t) \) instead of a constant current \( I \), we add to equation (4.2) a sufficiently high dimensionality to permit chaos to occur. Indeed, with the same parameters values used to produce hysteresis, and \( I_0 = 0.075nA, A = 0.007nA \) and \( \omega = 264.6Hz \), a chaotic spike train will be generated, as shown in figure 4.6.

![Figure 4.6: Chaotic spike train Phase plane showing the limit cycle and nullclines for the human neuron for \( I = 0 \) (left) and \( I = 0.6\pi A \) (right).](image)

The presence of chaos in neural systems arise some important questions. For instance, we must consider that the sensitivity to initial conditions implies lack of long-term predictability, so that greater descriptive detail in a chaotic neural system will not necessarily lead to greater predictive capability. At the level of animal behavior this unpredictability can be of some advantage, as the prey fleeing from a predator may be using some adaptative application of neural chaos in motor control [Wil99]. In a more philosophic approach, since the cognitive functions are located in the cerebral cortex [KSJ00] and there may be chaotic neural subsystems, neural chaos may well explain unexpected human behavior. Indeed, there is already evidence of chaos in the monkey cortex and simulations have suggested that cortical networks with recurrent excitation or inhibition may exhibit chaos [vVS96].

4.5 Conclusions

From this brief discussion we can see that up to the present days, the usual application of the more detailed neuron models has been the simulation of real biological systems, which frequently display a nonlinear dynamical behavior. It is also worth noting that the elaboration and application
of these more detailed models to do *engineering* tasks is still in its beginning, and there is a lot of work to be done to make the present models more tractable.
Chapter 5

Practical Applications Using SNN

5.1 Introduction

Once we have the knowledge of the basic principles and concepts of the real neurons and their various mathematical models, it is now time to apply this knowledge to build a SNN to carry on some practical tasks. Therefore, the main objective of this chapter is to show how neurons can encode and process information based only on spikes. Although there are several works dealing with clustering with SNN [SW99, Jin04, KIKY03], the examples given here use the unsupervised clustering method suggested in [Hop95] and implemented in [Ruf98] and [BPK02], with some minor modifications and variations. All implementations were written in MATLAB scripts [HL03], some of which are listed in Appendix A.

Before building a SNN, however, we have to explore three important issues, i.e., information encoding, learning method and network architecture, which will be discussed in the first sections. After that we will use the SNN to carry on some clustering tasks.

5.2 Information Encoding

The first question that arises when dealing with spiking neurons is how neurons encode information in their spike trains, since we are specially interested in a method to translate an analog value into spikes [HMR02], so we can process this information in a SNN. This fundamental issue in neurophysiology remains still not completely solved [RWWdRvSB97] and is extensively treated in several publications [SKdRvSB98, AT02, YS99, AA99, Koc99, PP95].

Although a line dividing the various coding schemes cannot always be clearly drawn [DA01], it is possible to distinguish essentially three different approaches [MB98, GK02b] in a very rough categorization:
rate coding: the information is encoded in the firing rate of the neurons [Wil99].

temporal coding: the information is encoded by the timing of the spikes. [Ruf98, Boh04]

population coding: information is encoded by the activity of different pools (populations) of neurons, where a neuron may participate of several pools [Sni96, Ger99, GK02b].

Figure 5.1: Simple temporal encoding scheme for two analog variables: $x_1 = 3.5$ and $x_2 = 7.0$, with $x_3$ as the reference, firing always at 0 in a coding interval of 10ms.

A very simple temporal coding method, suggested in [Hop95, Maa97, MB98], is to code an analog variable directly in a finite time interval. For example, we can code values varying from 0 to 20 simply by choosing an interval of 10ms and converting the analog values directly in proportional delays inside this interval, so that an analog value of 9.0 would correspond to a delay of 4.5ms. In this case, the analog value is encoded in the time interval between two or more spikes, and a neuron with a fixed firing time is needed to serve as a reference. Figure 5.1 shows the output of three spiking neurons used to encode two analog variables. Without the reference neuron, the values 3.5 and 7.0 would be equal to the values 6.0 and 2.5, since both sets have the same interspike interval.

If we now fully connect these three input neurons to two output neurons, we will have a SNN like the one shown in figure 5.2, which is capable of correctly separating the two clusters shown in the right side of the figure. Although this is a too simple example, it is quite useful to demonstrate how real spiking neurons possibly work. The clustering here was made using only the axonal delays $\Delta_{ij}^{ax}$ (see Chapter 2) between the input and output neurons with all the weights equal to one.

In a SNN with $m$ input and $n$ output neurons, the center of the RBF-like output neuron $j$ is given by the vector $c_j = [c_{1,j}, c_{2,j}, \ldots, c_{m,j}]$, where $c_{i,j} = \max \{\Delta_{ij}^{ax} | 1 \leq i \leq m\} - \Delta_{ij}^{ax}$. Similarly, the input vectors are defined as $x = [x_1, x_2, \ldots, x_m]$, where $x_i = t_i - \min \{t_i | 1 \leq i \leq m\}$ and $t_i$ is the
firing time of each input neuron [Ruf98]. The SNN used here has an EPSP (see figure 2.4) with a time constant $\tau = 1.84 ms$ and a threshold $\theta = 2.04$. The lateral connection between the two output neurons is strongly inhibitory and will be explained in the following sections.

This encoding method can work well for a number of clusters less or equal to the number of dimensions, but its performance decreases when the number of clusters exceeds the number of input dimensions. The proposed solution for this problem [BPK02] implemented here uses an encoding method based on population coding [Sni96, dKvdV01], which distributes an input variable over multiple input neurons. By this method, the input variables are encoded with graded and overlapping activation functions, modeled as local receptive fields. Figure 5.3 shows the encoding of the value 0.3. In this example, assuming that the time unit is millisecond, the value 0.3 was encoded with six neurons by delaying the firing of neurons 1 (5.564ms), 2 (1.287ms), 2 (0.250ms), 4 (3.783ms) and 5 (7.741ms). Neuron 6 does not fire at all, since the delay is above 9ms and lays in the no firing zone. It is easy to see that values close to 0.3 will cause neurons 2 and 3 to fire earlier than the others, meaning that the better a neuron is stimulated, the nearer to $t = 0ms$ it will fire. A value up to $t = 9ms$ is assigned to the less stimulated neurons, and above this limit the neuron does not fire at all (see figure 5.4).

Biologically, we can think of the input neurons as if they were some sort of sensory system sending signals proportional to their excitation, defined by the Gaussian receptive fields. These neurons translate the sensory signals into delayed spikes and send them forward to the output neurons.

In this work, the encoding was made with the analog variables normalized in the interval [0,1]
Figure 5.3: Continuous input variable encoded by means of local receptive fields.

and the receptive fields equally distributed, with the centers of the first and the last receptive fields laying outside the coding interval [0,1], as shown in figure 5.3. There is another way to encode analog variables, very similar to the first, the only difference being that no center lays outside the coding interval and the width of the receptive fields is broader.

Both types of encoding can be simultaneously used, like in figure 5.5, to enhance the range of detectable detail and provide multiscale sensitivity [BPK02]. The width $\sigma$ and the centers $c_i$ are defined by equations (5.1) and (5.2), for the first and second types, respectively. Unless otherwise mentioned, the value of $\gamma$ used for the first type is 1.5 and 0.5 for the second.

\[
\sigma = \frac{1}{\gamma(m+1)} \quad c_i = \frac{i-1}{m-1} \tag{5.1}
\]

\[
\sigma = \frac{1}{\gamma(m-2)} \quad c_i = \frac{i-1.5}{m-2} \tag{5.2}
\]
5.3 Learning

The approach presented here implements the Hebbian reinforcement learning method [Heb49] through a winner-takes-all algorithm [BLdC00, Hay99], whose practical application in SNN is discussed in [Ruf98, BPK02, dQ02] and a more theoretical approach is presented in [GK02a, BSS93]. In a clustering task, the learning process consists mainly of adapting the time delays, so that each output neuron represents an RBF center. This goal is achieved using a learning window \(^1\) [KGvH99a, KGvH99b], which is defined as a function of the time interval \(\Delta t_{ij}\) between the firing times \(t_i\) and \(t_j\). This function controls the learning process by updating the weights based on this time difference, as shown in equation (5.3), where \(\Delta w_{ij}\) is the amount by which the weights \(w_{ij}\) are increased or decreased and \(\eta\) is the learning rate.

\[
\Delta w_{ij} = \eta L(\Delta t_{ij})
\]  

(5.3)

The learning function used here, shown in figure 5.6, is a Gaussian curve defined by the equation (5.4) [GK02a, KGvH99a, KGvH99b, LvH01]. It reinforces the synapse between neurons \(i\) and \(j\) if \(\Delta t_{ij} < \nu\), and depresses the synapse if \(\Delta t_{ij} > \nu\).

\[
L(\Delta t) = (1 + \beta) e^{\frac{(\Delta t - \alpha)^2}{2(\kappa - 1)}} - \beta
\]

(5.4)

\(^1\)The learning window is also called learning function and both terms will be used here with the same meaning.
Figure 5.5: The two ways to encode continuous input variable by means of local receptive fields. The dotted lines are wide receptive fields of the second type, with $\lambda = 0.5$

where

$$\kappa = 1 - \frac{\nu^2}{2 \ln \frac{\beta}{1+\beta}}$$

The learning window is defined by the following parameters:

- $\nu$: this parameter, called here neighborhood, determines the width of the learning window where it crosses the zero line and affects the range of $\Delta t_{ij}$, inside which the weights are increased. Inside the neighborhood the weights are increased, otherwise they are decreased.

- $\beta$: this parameter determines the amount by which the weights will be reduced and corresponds to the part of the curve laying outside the neighborhood and bellow the zero line.

- $\alpha$: because of the time constant $\tau$ of the EPSP (see figure 2.4), a neuron $i$ firing exactly with $j$ does not contribute to the firing of $j$, so the learning window must be shifted slightly to consider this time interval and to avoid reinforcing synapses that do not stimulate $j$.

5.4 Network Architecture

The architecture adopted here is almost the same as in a rate-based neural network (e.g. a feed-forward network with the Perceptron model [Ros58]), with the neurons of the preceding layer fully connected to the neurons of the next layer. The first layer is composed by the receptive fields neurons and the second, by the output neurons, also called RBF neurons [BPK02]. If other layers
are added, the neurons of the previous layer will act as the receptive fields of the next layer.

A first difference in relation to the rate-based neural networks is that all output neurons are laterally connected to each other by means of a strongly inhibitory synapse. The purpose of this inhibitory synapse is to disable all other neurons after the first has fired, performing a winner-takes-all mechanism.

Another difference is the multiple synapse approach. The existence of multiple synapses is biologically plausible (e.g. [WZR98]), since in brain areas like the neocortex a single presynaptic axon makes several independent contacts with the postsynaptic neuron. The practical aspects of this theory has been recently discussed and neural computation using this approach has been already demonstrated [MZ99, NMZ01, NSD02, WF02].

Instead of a single synapse, with its specific delay and weight, this synapse model consists of many sub-synapses, each one with its own weight and delay, as shown in figure 5.7. Although the final effect is like the one of a single synapse (see figure 5.8), the use of multiple sinapses enables an adequate delay selection using the learning rule presented in the previous section. For each multiple synapse connecting neuron $i$ to neuron $j$, with $s$ sub-synapses, the resulting PSP is given by equation (5.5).

$$ u_{ij}(t) = \sum_{k=1}^{s} w_{ij}^k e(t - t_i - d^k) \quad (5.5) $$

Where $d^k$ has the same effect of the axonal delay represented by $\Delta_{ij}^{ax}$ in figure 2.4. The total contribution of all presynaptic neurons is then given by equation (5.6).
Figure 5.7: A single multiple synapse. The final EPSP is the summation of all potentials produced by the sub-synapses.

\[ u_j(t) = \sum_{i \in f_j} \sum_{k=1}^{s} w_{ij}^k \epsilon(t - t_i - d^k) \]  \hspace{1cm} (5.6)

The delays \( d^k \) are fixed for all sub-synapse \( k \), varying from zero in 1ms fixed intervals. The most used arrangements have 12 to 16 sub-synapses, with delays up to 15ms. These parameters are chosen experimentally, to produce the best results. The neuron model implemented is the \( SRM_0 \) [GK02b], with a strictly excitatory PSP modeled by equation (2.3), with \( \tau = 3\text{ms} \). A leaky integrate-and-fire neuron could also be used with no modification in the network architecture.

It is worth mentioning that there is a fixed return delay [Ruf98] that should also to be included to take into account the effect of the feedback signaling from the membrane to the synapse. In our implementations we ignored this delay [BPK02]. Otherwise, we would have simply to add a new time delay to take into account this time interval.

With this architecture, the resulting PSP is the temporal summation of the PSP produced by each sub-synapse. In figure 5.8 we show the EPSP produced by the sub-synapses connecting the input neuron 6 to the output neuron 1 in the SNN used to separate five clusters, discussed in the next section (see also input neuron 6 in the output neuron 1 graph in figure 5.10)
5.5 Applying a SNN to a clustering Task

The first example developed here demonstrates the use of a SNN in a clustering task by correctly separating five bi-dimensional randomly generated clusters. The SNN uses the architecture explained in the previous section, with eight neurons encoding the inputs of each dimension, using a time interval of 10ms. Each cluster is represented by one of the five output neurons.

With this configuration we have $\Delta t_{ij} = t_i + d^k - t_j$ and the learning rule to be applied for updating all weights in all sub-synapses is given by equation (5.7).

$$\Delta w_{ij}^k = \eta L(t_i + d^k - t_j)$$ (5.7)

Since with population coding some of the neurons will not fire, these neurons must also be penalized, as well as the neurons firing in a time interval outside the neighborhood $v$. The strategy applied here was to adjust the weights of these neurons with $\Delta w_{ij}^k = -\eta$. It is also important to note that the values of the weights were limited to the interval $[0,1]$, i.e., $0 \leq w_{ij}^k \leq 1$, to avoid instability of the SNN.

The clusters were randomly generated with a total of 50 points each and, also randomly, 25 points were separated for training and the remaining points were used for validation. Each training epoch consisted of the presentation of the points and the weight updating according to equation (5.7). The experiments started with learning rates as low as $\eta = 0.0025$ and a number of epochs above 100, but the results demonstrated that higher learning rates and less epochs could be used without altering the correct separation of the clusters. In figure 5.9 it is possible to ob-
Figure 5.10: Five clusters correctly separated by a SNN with 8 neurons encoding each dimension. The resulting weights are shown for all five clusters. The SNN used 13 sub-synapses, with delays from 0 to 12ms and a learning rate $\eta = 0.35$. 

53
Also important to the network performance are the limiar $\theta$ and the neighborhood $\nu$, and if these parameters are not finely adjusted, the network will respond wrongly or will not respond at all to some of the inputs. The values used here were $\theta = 12$ and $\nu = 5$. The final results and corresponding weight configuration are shown in figure 5.10.

Other tests were also successfully made, with up to ten clusters, with minor necessary adjustments in the parameters. For the ten 50-points clusters 12 neurons were used to encode each dimension. The main parameters used are listed in table 5.1.

In order to demonstrate the capabilities of the present method with more complex clusters, we used a 256 x 256 pixels, 3-band RGB satellite image, where each color pixel is defined by a three dimensional vector with dimensions varying from 0 to 255. The network was trained using 32,769 randomly chosen points (of a total of the 65,536 points) to separate four clusters. The results of the clustering with the SNN are shown in figure 5.11 with the original image and the results of the same clustering using a self-organized map – SOM [BLdC00, Hay99] and the K-means algorithm. The SNN used only six neurons to encode each dimension. As expected, the clustering with the SNN showed the highest computational cost, followed by the SOM implementation, using a Matlab toolbox routine.

<table>
<thead>
<tr>
<th>Example</th>
<th>Clusters</th>
<th>Dimensions</th>
<th>Points</th>
<th>$\theta$</th>
<th>$\nu$</th>
<th>Input</th>
<th>Coding</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>2</td>
<td>250</td>
<td>12</td>
<td>5</td>
<td>16</td>
<td>10ms</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>2</td>
<td>500</td>
<td>12</td>
<td>8</td>
<td>24</td>
<td>10ms</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
<td>65536</td>
<td>9</td>
<td>5</td>
<td>18</td>
<td>20ms</td>
</tr>
</tbody>
</table>

5.6 A Novel Approach

Although the existence of multiple synapses may be biologically plausible (e.g. [WZR98]), we can not help thinking of this case as being more a particularity than a rule. As a matter of fact, it is clear that the multiple synapse approach discussed here is far more a delay adaptation mechanism than a solution to abide by biological plausibility. As a consequence, it is natural to suppose that there must be a way for single-synapse neurons to do the same sort of task of the multiple synapse neurons.

To test and demonstrate this novel approach, it is necessary to have a single EPSP equivalent to that produced by each multiple synapse assembly (figure 5.8). This equivalent EPSP was

---

2 Extract of a IKONOS II scene, PSM mode, RGB bands 3-2-1 and Pan, natural colors, 1m resolution.
Figure 5.11: Results of an image clustering task. Clockwise from top: the original image, clustering by the k-means algorithm, clustering by the SOM (using Matlab) and clustering by the SNN.

calculated using two methods, as shown in figure 5.12.

- In the first method, the equivalent EPSP was calculated using variable time constants ($r_{ij}$) equal to the instants of the maximum of the original EPSP, and a null axonal delay ($\Delta_{ij} = 0$).

- In the second method the equivalent EPSP was calculated using a fixed time constant ($\tau = 3$) and a variable axonal delay ($\Delta_{ij}^{\tau}$), so that the maximum of this EPSP coincides with the maximum of the original EPSP.

In both methods equation (2.3) was adopted for the equivalent EPSP, so that its maximum amplitude is the same as that of the original EPSP, occurring at the same time. The maximum value was used as the weight (figure 5.12).
Figure 5.12: Two single-synapse EPSPs (dashed lines) equivalent to that produced by a multiple synapse assembly (solid lines). The figure shows the synapse between input neuron 1 and output neuron 2, in the five clusters experiment. At left, we have the equivalent EPSP using variable time constants and a null axonal delay (first method). At right we have the equivalent EPSP using a fixed time constant \( \tau = 3 \) and a variable axonal delay (second method). Equation (2.3) was used in both cases.

Taking the five clusters example, for each of the 16 resultant EPSP we wrote down the maximum value and its time \( (w \text{ and } t_w \text{ in figure 5.12}) \), and calculated the single-synapse equivalent EPSP by both methods, and then repeated the clustering task with the same temporal encoded input. As expected, these new single-synapse SNNs presented the same results as the multiple synapse counterpart, except for the second method, where we observed five clustering errors (2%). The matrices for the weights, time constants and delays are shown below.

These results demonstrates the proposed approach but, albeit this successful outcome, a main question remains unanswered: how unsupervised reinforcement learning should be applied to such a SNN?

Despite some few references on delay adaptation may be found [EPE+99, EPE+00, LvH01, LKvH02, SZK99, YS98], most of these works only present a general approach, and none of them deal with the specific kind of problem we have here.

As a first trial to answer this remaining question, considering the results of the first method, we adopted the simultaneous and independent adaptation of the weights and time constants in a SNN of single synapses. For the weight adaptation we used the same learning function as in the previous examples, and for the adaptation of the time constants we used the learning function mentioned in [IGE04, GK02b], defined by equation (5.8) and shown in figure 5.13.
\[ \begin{bmatrix}
0 & 3.524 & 0.042 & 0 & 0 \\
0 & 6.980 & 6.499 & 0 & 0 \\
0 & 4.330 & 5.861 & 0 & 4.692 \\
0 & 0 & 0 & 0 & 6.709 \\
0.527 & 0 & 0 & 0 & 3.158 \\
6.281 & 0 & 0 & 4.114 & 0 \\
6.084 & 0 & 0 & 6.916 & 0 \\
0 & 0 & 0 & 3.166 & 0 \\
0 & 4.026 & 0 & 0 & 0 \\
2.463 & 6.983 & 0 & 0 & 0 \\
6.697 & 3.655 & 0 & 0 & 4.712 \\
4.903 & 0 & 0 & 0 & 6.983 \\
0 & 0 & 0 & 4.414 & 2.303 \\
0 & 0 & 4.076 & 6.964 & 0 \\
0 & 0 & 6.812 & 2.982 & 0 \\
0 & 0 & 3.692 & 0 & 0
\end{bmatrix} \]

\[ \begin{bmatrix}
0 & 4.8 & 3 & 0 & 0 \\
0 & 9.8 & 8.8 & 0 & 0 \\
0 & 5.5 & 7.6 & 0 & 6.2 \\
0 & 0 & 0 & 0 & 9.4 \\
3.8 & 0 & 0 & 0 & 4.6 \\
8.3 & 0 & 0 & 5.6 & 0 \\
8 & 0 & 0 & 9.8 & 0 \\
0 & 0 & 0 & 5.3 & 0 \\
0 & 5.3 & 0 & 0 & 0 \\
4.4 & 9.8 & 0 & 0 & 0 \\
9.4 & 4.9 & 0 & 0 & 6 \\
6.5 & 0 & 0 & 0 & 9.8 \\
0 & 0 & 0 & 5.8 & 3.9 \\
0 & 0 & 5.4 & 9.8 & 0 \\
0 & 0 & 9.6 & 4.5 & 0 \\
0 & 0 & 4.9 & 0 & 0
\end{bmatrix} \]

\[ \begin{bmatrix}
0 & 1.8 & 0 & 0 & 0 \\
0 & 6.8 & 5.8 & 0 & 0 \\
0 & 2.5 & 4.6 & 0 & 3.2 \\
0 & 0 & 0 & 0 & 6.4 \\
0.8 & 0 & 0 & 0 & 1.6 \\
5.3 & 0 & 0 & 2.6 & 0 \\
5 & 0 & 0 & 6.8 & 0 \\
0 & 0 & 0 & 2.3 & 0 \\
0 & 2.3 & 0 & 0 & 0 \\
1.4 & 6.8 & 0 & 0 & 0 \\
6.4 & 1.9 & 0 & 0 & 3 \\
3.5 & 0 & 0 & 0 & 6.8 \\
0 & 0 & 0 & 2.8 & 0.9 \\
0 & 0 & 2.4 & 6.8 & 0 \\
0 & 0 & 6.6 & 1.5 & 0 \\
0 & 0 & 1.9 & 0 & 0
\end{bmatrix} \]

\[
L(\Delta t) = \begin{cases} 
-\left(\frac{\Delta t + \alpha}{\tau}\right) & \text{for } \Delta t \geq 0 \\
-\left(\frac{\Delta t + \alpha}{\tau}\right)^{-1} & \text{for } \Delta t < 0 
\end{cases}
\]  
\hfill (5.8)

Although several variations and parameter set were used, this approach does not produced any acceptable results in almost all tentatives, suggesting that the independent adaptation of weights and time constants is not a good solution for the problem in perspective. As a matter of fact, the results of this approach showed more errors than those obtained just using a fixed value for the time constants and adapting only the weights. In this last case a few tentatives produced a clustering without errors.

As a second trial, in order to find experimentally a relationship between weights and time constants, we plotted a map of the normalized weight and time constants values, for the five and ten clusters experiments. These maps, shown in figure 5.14, suggest that there may be indeed some relationship. A curve fitting procedure yielded equations (5.9) and (5.10), for the five and ten clusters cases, respectively.
Figure 5.13: Alternative learning function applied to delay adaptation in a single synapse approach.

\[ \tau(w) = 0.2452e^{-0.07922w} + 0.09115e^{2.149w} \]  

(5.9)

\[ \tau(w) = 0.2716e^{-0.8521w} + 0.07901e^{2.413w} \]  

(5.10)

Both experiments were then repeated 50 times each, using equations (5.9) and (5.10) to calculate the time constants as a function of the weights, which were adapted with the same procedures of the multiple synapse method, i.e., using equations (5.3) and (5.4). Around 80% of the tentatives showed no errors, indicating that there must be indeed a relationship connecting weights and time constants. The results also pointed out the need of a deeper theoretical and mathematical analysis to explore the complexity of the problem.

5.7 Conclusions

Together with some examples of the application of a multiple synapse SNN, we presented here a novel architecture with single synapses and applied it successfully to do clustering tasks.

Since the objective of the learning process is to approximate the firing times of all the neurons related to the same cluster, it is quite obvious that a neuron less stimulated (large \( \Delta t \) and thus, low weight) must have also a lower time constant, so it can fire faster and compensate for the large
Figure 5.14: Maps for the five (left) and ten (right) clustering experiments to find experimentally a relationship between weights and time constants.

\( \Delta t \). Similarly, a more stimulated neuron (small \( \Delta t \) and thus, high weight) must have also a higher time constant, so it can fire slower and compensate for the small \( \Delta t \).

As already mentioned in the previous section, all the experimental results obtained in the development this chapter indicate that the simultaneous adaptation of weights and time constants (or axonal delays) must be submitted to a far more extensive theoretical analysis. Given the high complexity of the problem, it is not encompassed by the scope of the present work, and hence should be left to a further work.
Chapter 6

Conclusions

6.1 Outlook

In this work, as was our main target, we tried to summarize the foundations of spiking neurons and to emphasize the importance of considering this new paradigm when building neural networks. To accomplish this task we first presented the basic components of the nervous systems, giving a glimpse of all their underlaying complexity. Following, we then discussed some of the mathematical models that allow us to simulate these systems, from the simpler to the more complex ones. After the exposition of the theoretical basis, we briefly discussed the dynamics of real neural processes, with a few examples of application of the models previously described. It was also presented a practical applications of a neural network, built with more biologically inspired neurons, to perform what we could call engineering tasks. In this application we demonstrate how analog values can be temporally encoded and how a network can learn using this temporal code.

Even with these very short steps towards the realm of neuroscience, it is not difficult to realize how intricate things can get if we try to descend deeper into the details of neural simulation. However, this apparent difficulty should rather be regarded as an opportunity to use spike-timing as an additional variable in the information processing by neural networks [Boh04].

The construction presented here and also other types of SNN seem to be suitable for hardware implementation [MB98, SHY00, SSW+02, KA05]. There are already several implementations of Hebbian learning with spiking neurons [Häf00, LMB00]. Real spiking neural networks are known to be very fast [Ger99, GK02b] and the single synapse approach discussed in Chapter 5 could represent a significant reduction in computation time if compared to the multiple synapse approach. It is worth noting that other learning methods have been used with SNN, like supervised learning [XE01, BKP02]. We can find also SNN applications in other fields, e.g., [MB00, SC99].

In the field of real neural systems and cognitive sciences, the advance in computer power has
ensued a trend to simulate large populations [IGE04, MG02, Vol04] to study their behavior. The elaboration and application of the more detailed neuron models to do engineering tasks is still in its beginning, and there is a lot of work to be done to make the present models more tractable.

Besides the extensive review of spiking neurons, although a definitive solution for the training problem is out of the scope of this work, we believe to be contributing with a new direction of research by demonstrating that the single synapse approach can be adopted with the advantage of reducing the computational costs.

6.2 Future Works

Focusing specially in neural modeling and related applications, this work has only slightly touched the wide field of neuroscience, and so, it is only a first step to be followed by many others, toward different directions, representing the broad range of research possibilities for future development. For instance, one could go deeper into Neurophysiology to uncover the intricate structure of the nervous systems at the molecular level, enabling the use of proteins and other molecules for information-processing, computation in nonlinear media, computers based on physical reaction-diffusion systems found in chemical media, DNA computing, bioelectronics and protein-based optical computing, and biosensors.

Despite this broad range of possibilities, we will limit ourselves to only two research choices, directly related with the subjects discussed here: neural code and the single synapse approach suggested in Chapter 5. Although the code scheme used here to encode analog variables into precisely spike timing worked well in our implementation, this subject is worth of further investigation in order to determine how analog inputs are encoded in real nervous systems. It will not be an easy task and this question still remains unsolved [GK02b].

For the single synapse approach proposed here, all the experimental results obtained indicate that the simultaneous adaptation of weights and time constants (or axonal delays, or both) should be submitted to a far more extensive theoretical analysis. We believe that axonal delays, time constants and weights are closely interrelated and, once a dependence relationship is determined, multiobjective optimization techniques could be employed to define a learning rule.
Appendix A
Matlab Scripts

We present here the final version of the Matlab 6 scripts [HL03] used in the simulations of Chapter 5. The listed scripts were used to the five clusters experiment and can be applied to any number of cluster with little modifications.

A.1 Temporal encoding - Section 5.2
A.1.1 Linear encoding

function aus = kodieren_ln(ein, maxd, rho, plt)
% Funcao aus = kodieren_ln(ein, maxd, rho, plt)
% Descricao: Codifica valores analogicos transformando-os diretamente
% em atrasos dentro do intervalo de codificacao. A quantidade
% de neuronios e igual a quantidade de entradas (dimensoes).
% ein = matriz i x j com os i exemplos para treinamento e
% validacao e as j entradas analogicas (dimensoes).
% Cada linha de ein e um conjunto de variaveis
% analogicas de entrada.
% aus = matriz m x n com os atrasos de todas as entradas
% analogicas codificadas, onde m e a quantidade de
% exemplos (m = i) e n e a quantidade total de
% neuronios de entrada usados para codificar as
% entradas analoticas. Cada linha de aus contem os
% atrasos dos n neuronios de entrada (n = rf x j).
% maxd = intervalo de codificacao.
% rho = resolucao de tempo.
% plt = 1 - plota a saida
% 0 - nao plota a saida
%
%---------------------------------
% Ultima alteracao em 09/04/2005 - RCBerredo
%---------------------------------
% [t_cj,t_in]=size(ein); % conjuntos (t_cj) e entradas analogicas (t_in)
% % Normaliza entradas no intervalo (0,1)
% for i=1:t_in
%   range=max(ein(:,i))-min(ein(:,i)); % faixa de valores da variavel
%   ein(:,i)=(ein(:,i)-min(ein(:,i)))./range; % variaveis norm. de 0 a 1
% end

62
% Calcula atrasos e monta saida
% aus=ein.*maxd;
% aus=round((ein.*maxd)./rho).*rho;
%
% Plota os resultados
if plt=1
    figure;
    for i=1:t_cj
        for j=1:t_in
            subplot(t_in,1,j); hold on;
            H_line=plot([aus(i,j) aus(i,j)],[0 1],’b’);
            set(H_line,’LineWidth’,1);
            axis([0 maxd 0 2]);
            Ha_ax=gca;
            ylabel(sprintf(’%lg’,j),’FontSize’,8);
            set(Ha_ax,’YTick’,[]);
            set(Ha_ax,’XTick’,[]);
            %set(Ha_ax,’XTick’,[0 maxd*0.2 maxd*0.4 maxd*0.6 maxd*0.8 maxd]);
            set(Ha_ax,’XGrid’,’on’,’XTickLabel’,’ ’);
            if j==t_in
                set(Ha_ax,’XTickLabelMode’,’auto’);
                xlabel(’t (ms)’,’FontSize’,12);
            end
            if j==1
                title(sprintf(’Classes: %lg - Entrainadas: %lg - ...
                        Exemplos: %lg’, t_in,t_in,t_cj),’FontSize’,14);
            end
        end
    end
elseif plt=0
    error(’Parametro plt invalido: tem que ser 0 ou 1’);
end

A.1.2 Receptive fields encoding

function aus = kodieren_rf(ein, rf, maxd, sigma, f_cut, rho, typ, plt)
    % Funcao aus = kodieren_rf(ein, rf, maxd, sigma, f_cut, rho, typ, plt)
    % Descricao: Codifica valores analógicos através de múltiplos campos receptivos fixos.
    %
    ein = matriz i x j com os i exemplos para treinamento e validacao e as j entradas analógicas (dimensiones).
    % Cada linha de ein e um conjunto de variaveis analógicas de entrada.
    aus = matriz m x n com os atrasos de todas as entradas analógicas codificadas, onde m e a quantidade de exemplos (m = i) e n e a quantidade total de neuronios de entrada usados para codificar as entradas analógicas. Cada linha de aus contem os atrasos dos n neuronios de entrada (n = rf x j).
    rf = matriz linha com a quantidade de neuronios usados na codificacao de cada entrada (dimenso).
    maxd = intervalo de codificacao.
    sigma = matriz linha com a largura de cada grupo de campos receptivos. Recomenda-se o sigma = 1/(1,5(m-2)) para o tipo 0 e sigma = 1/(1,5(m+1)) para o tipo 1.
    f_cut = limite de disparo dos campos receptivos.
% rho = resolução de tempo.
% typ = 1 - nenhum neurônio está localizado fora da faixa de
% valores a serem codificados.
% 0 - dois neurônios estão localizados fora da faixa de
% valores a serem codificados, sendo um de cada lado.
% plt = 1 - plota a saída.
% 0 - não plota a saída.

% Ultima alteração em 17/05/2005 - RCBerredo
%--------------------------------------------------------
t_neur=sum(rf); % total de neurônios
[t_cj,t_in]=size(ein); % conjuntos (t_cj) e entradas analóg. (t_in)
aus=zeros(t_cj,t_neur); % cria matriz de saída
%sigma=0.233./rf; % valor para que o atraso max < 0.9*dmax

% Processa as entradas
n=0; % controle de colunas da matriz de saída (aus)
for i=1:t_in
    ct=zeros(1,rf(i)); % centros das curvas
    % Verifica quantidade de neurônios
    if rf(i)<3
        error('Quantidade de campos receptivos tem que ser maior que 3');
    end
    % Normaliza entradas no intervalo (0,1)
    range=max(ein(:,i))-min(ein(:,i)); % faixa de valores da variável
    ein(:,i)=(ein(:,i)-min(ein(:,i)))./range; % variáveis norm. de 0 a 1
    % Define os centros das curvas
    if typ==1
        cdis=1/rf(i); % distância entre os centros
        for j=1:rf(i)
            ct(j)=(j-0.5)*cdis;
        end
    elseif typ==0
        cdis=1/(rf(i)-2); % distância entre os centros
        for j=1:rf(i)
            ct(j)=(j-1.5)*cdis;
        end
    else
        error('Parametro typ invalido: tem que ser 0 ou 1');
    end
    % Calcula atrasos e monta saída
    for j=1:t_cj
        for k=1:rf(i)
            aux=maxd-maxd.*exp(-(ein(j,i)-ct(k))^2)/(2.*sigma(i)^2));
            aus(j,k+n)=round(aux*(1/rho))\*rho; % arredonda
            if aus(j,k+n)>maxd*f_cut % se o neurônio não dispara
                aus(j,k+n)=-1; % o atraso é -1
            end
        end
    end
    n=n+rf(i); % controle de colunas da matriz de saída (aus)
end
% Plota os resultados
if plt==1
    figure;
for i=1:t_cj
    for j=1:t_neur
        subplot(t_neur,1,j); hold on;
        H_line=plot([aus(i,j) aus(i,j)],[0 1],’b’);
        set(H_line,’LineWidth’,1);
        axis([0 maxd 0 2]);
        Ha_ax=gca;
        ylabel(sprintf(’%ig’),j,’FontSize’,8);
        set(Ha_ax,’XTick’,[]);
        set(Ha_ax,’XTick’,[0 maxd]);
        set(Ha_ax,’XTick’,[0 maxd*0.2 maxd*0.4 maxd*0.6 maxd*0.8 maxd]);
        set(Ha_ax,’XTickLabel’,’’);
        if j==t_neur
            set(Ha_ax,’XTickLabelMode’,’auto’);
            xlabel(’t (ms),’,’FontSize’,12);
        end
        if j==1
            title(sprintf(’Classes: %ig – Entradas: %ig – ...
            Exemplos: %ig’, t_in,t_neur,t_cj),’FontSize’,14);
        end
    end
end
figure; hold on;
x=0:0.01:1;
for i=1:rf(end)
    y=maxd.*exp(-(x-ct(i)).^2/(2*sigma(end).^2));
    plot(x,y);
end
plot([0 1],[maxd*(1-f_cut) maxd*(1-f_cut)],’r’);
elseif plt==0
    error(’Parametro plt invalido: tem que ser 0 ou 1’);
end

A.1.3 Simple clustering example

% cl_spk_2c - teste de classificacao com SNN
% % Ultima alteracao 05/06/2005 - RCEberredo
% % Usa 2 conjuntos com distribuicao gaussiana com media zero.
% % Cada conjunto tem 50 pontos.
% % O metodo usado foi o de adaptar, de forma supervisionada,
% % os arcos axonais. Os pesos foram mantidos iguais a 1.
% % Classes (saidas): 2
% % Codificacao: Linear direta com um neuronio de referencia
% % Conj. de Campos Receptivos: 0
% % Tipo de Sinapses: simples
% %
% % Prepara Entrada
% aux1=dimread(’cjs_spk2a_orig.txt’); % dados de entrada
% %
% % Define Parametros Iniciais
% tam=size(aux1,1); % quantidade de conjuntos
% in_neu=size(aux1,2)+1; % qtd de neuronios de entrada
% out_neu=2; % qtd de neuronios na saida (classes)
% rho=0.1; % resolucao de tempo
% tau=1.84; % 1.93; % cte de tempo do EPSP

65
maxd=10;  % intervalo de codificacao
tmax=30;  % maximo intervalo de calculo
teta=2.04; %2.35;

% Treina a rede
tra=50;  % tamanho do conjunto de treinamento
trb=(tam/2)+tra;
aus=kodieren_ln(aux1,maxd,rho,0);  % codifica entradas
%ca=[2.0 2.5];%ca=[7.8 6.5];
ca=round((sum(aus{1:tra,:}))/tra)./rho).*rho;  % calcula media da classe 1
%cb=[8.2 6.4];%cb=[3.3 2];
cb=round((sum(aus{51:trb,:}))/tra)./rho).*rho;  % calcula media da classe 2
d=maxd-[ca{1} cb{1}]; ca{2} cb{2}]; maxd maxd];  % atrasos dos centros
w=[1 1 1 1 1;1 1 1 1 1]; %teta; %1 1.42;1 1.38;1 1];

% Testa a rede
class=zeros(1,tam);  % cria tabela de classes
aux1=dlmread('cyspkn2a.txt');  % dados de entrada
aus=kodieren_ln(aux1,maxd,rho,0);  % codifica entradas
aus=[aus ones(tam,1).*maxd];  % acrescenta referencia
for k=1:tam
    t=0; %min(aus{1,:});
    neu=0;
    while neu==0 & t<tmax
        out=zeros(1,out_neu);
        for j=1:out_neu
            for i=1:in_neu
                dt=t-aus{1,k,i};
                sai=w{i,j}*((dt-d{i,j}))/tau)*exp(1-((dt-d{i,j}))/tau);
                if sai<0
                    sai=0;
                end
            end
            out{1}=out{1}+sai;
        end
        %subplot(out_neu,1,j); hold on;
        %plot(t,out{1},'.','t',teta,'r. '); drawnow;
        end
        if max(out)>teta
            neu=find(out==max(out));
            inst=t;
        end
        t=t+rho;
    end
    if size(neu,2)>1  % se deu empate...
        neu=0;  % classe = 0
    end
    if neu==0  % se acabou limite de tempo
        class(k)=0;  % classe = 0
    end
    class(k)=neu;  % guarda a classe
    %pause
    %h=gcf;  % pega hadle da figura
    %close(h);  %fecha figura
end

% Plota resultado do teste
ptos={‘+’k’ ‘ok’ ‘or’ ‘+’b’ ‘xk’ ‘sr’ ‘db’ ...
    ‘,’k’ ‘*’r’ ‘ob’ ‘+’k’ ‘xk’ ‘sb’ ‘dk’ ...
    ‘,’r’ ‘*’b’ ‘ok’ ‘+’r’ ‘xb’ ’sk’ ‘dr’};
figure; hold on;
for i=1:out_neu
    a=find(class==i);
    for n=1:size(a,2)
        plot(aux1(a(n),1),aux1(a(n),2),char(ptos(i)));
    end
end
a=find(class==0);
for n=1:size(a,2)
    plot(aux1(a(n),1),aux1(a(n),2),char(ptos(out_neu+1)));
end

%function plot(a1, p1, px, py, teta)
% subplot(a1,1,1); hold on;
% plot(px,py,'.',px,teta,'r.'); drawnow;

A.2 Multiple synapse architecture - Section 5.5

A.2.1 Clustering

% cl_spk_5a - teste de classificação com SNN
% % Última alteração 19/07/2005 - RCBerredo
% % Usa 5 conjuntos com distribuição gaussiana com média zero.
% % Cada conjunto tem 50 pontos.
% % Classes (saídas): 5
% % Codificação: Campos receptivos fixos
% % Conj. de Campos Receptivos: 1
% % Tipo de Sinapses: multiplas
% %
% % Prepara Entrada
clear all;
aux1=dlmread('cj_spk5.txt'); % dados de entrada
% % Define Parâmetros Iniciais
out_neu=5; % quantidade de neurônios na saída (classes)
rho=0.1; % resolução de tempo
conj=125; % quantidade de conjuntos de treinamento
tau=3; % cte de tempo do EPSP
rf=[8 8]; % quantidade de neurônios por entrada
sigma=[1/(1.5*(rf(1)-2)) 1/(1.5*(rf(2)-2))]; % largura dos campos recept.
f_cut=0.9; % intervalo de codificação
% % Define Parâmetros de Aprendizado
d=[0 1 2 3 4 5 6 7 8 9 10 11 12]; % atrasos das sub-sinapses
w_max=1; w_min=0; % pesos máximo e mínimo
max_epoch=3; % no. máximo de épocas
t_max=30; % tempo máximo de treinamento
eta=0.35; % taxa de aprendizado
beta=0.2; % cte para janela de aprendizado
nu=5.0; % vizinhança
dx=2.3; % deslocamento da janela de aprendizado
% p/ esquerda (+) ou direita (-)

67
% Calcula Parametros
kapa=1-(nu^2)/(2*log(beta/(beta+1))); % cte para janela de aprendizado
in_neu=sum(rf); % qtd de neurônios na entrada
ssin=size(d,2); % qtd de sub-sinapses
%teta=1.5*(in_neu/out_neu)*(w_max-w_min)*ssin/2;
teta=12; % valor do limiar

% Inicializa Peso
w=zeros(in_neu, out_neu, ssin);
for i=1:in_neu
    for j=1:out_neu
        w(i,j,:)=(w_min+rand(1,ssin).*w_max)*teta;
    end
end

% Inicia Treinamento
tic;
aus=kodieren_rf(aux1, rf, maxd, sigma, f_out, rho, 0, 0); % codifica entrada
ctrl_1=1; % controle do loop
delta_T=zeros(1,1,ssin);
h=waitbar(ctrl_1/max_epoch, sprintf('Executando %d de %d iteracoes', ...
    ctrl_1, max_epoch));
%show_w(w);
f_times=zeros(2,1);
while ctrl_1<max_epoch
    for z=1:conj
        tr=aus(z,:);
        [neu, inst]=wer_schiehst(tr, t_max, w, d, tau, rho, teta, 0);
        f_times=[f_times [neu inst]];
        if neu~0 & size(neu,2)==1
            for i=1:in_neu
                if tr(i)==-1
                    w(i, neu, :)=w(i, neu, :) + eta;
                else
                    delta_T(1,1,:)=tr(i)+d-inst; % calcula delta_T
                    w(i, neu, :) = w(i, neu, :) + eta .^ ((1+beta) .^... 
                        exp(-(delta_T+2.3)^2)/(2*kapa-2))-beta;
                end
            ndx=find(w(i, neu, :)<w_min); % garante limites
            for u=1:size(ndx,1)
                w(i, neu, ndx(u))=w_min;
            end
            ndx=find(w(i, neu, :)>w_max); % garante limites
            for u=1:size(ndx,1)
                w(i, neu, ndx(u))=w_max;
            end
        end
    end
    ctrl_1=ctrl_1+1; % incrementa contador de epocas
    waitbar(ctrl_1/max_epoch, h, sprintf('Executando %d de %d iteracoes', ... 
        ctrl_1, max_epoch));
teta=teta+(0.3*teta)/max_epoch;
end
end
close(h);
toc;

% Testa a rede
[conj, in_neu] = size(aus);
class = zeros(1, conj); % cria tabela de classes
for i = 1: conj
    tr = aus(i, :); % conjunto de teste
    [neu, inst] = wer_schiefst(tr, t_max, w, d, tau, rho, teta, 0); % quem disparou?
    if size(neu, 2) > 1 % se deu empate...
        neu = 0; % classe = 0
    end
    class(i) = neu; % guarda a classe
end

% Plota resultado do teste
ptos = ['b' 'k' 'or' 'b' 'xk' 'sr' 'ob'...
      'b' 'k' 'or' 'b' 'xk' 'sr' 'ob'...
      'b' 'k' 'or' 'b' 'xk' 'sr' 'ob'];
figure; hold on;
for i = 1: out_neu
    a = find(class == i);
    for n = 1: size(a, 2)
        plot(aux1(a(n), 1), aux1(a(n), 2), char(ptos(i)));
    end
end
a = find(class == 0);
for n = 1: size(a, 2)
    plot(aux1(a(n), 1), aux1(a(n), 2), char(ptos(out_neu + 1)));
end

A.2.2 Image clustering - K-means

% cl_img_kmeans - Classifica imagem com k-means
% o nome do arquivo com a imagem e 'original_corte.jpg'
% este script depende da função Kmeans_var()
% Ultima alteração 24/07/2005 - RCBerredo
% clear all
% le o arquivo e transforma de 256 x 256 x 3 para 65536 x 3
imag = double(imread('original_corte.jpg')); % transforma em double para poder processar
[x1 y1 z1] = size(imag); % pega dimensões
% transforma array tri para bi-dimensional x1 x y1 x z1 -> (x1 * y1) x z1
imagx = zeros((x1 * y1), z1); conta = 1; % cria área de saida
for i = 1:x1
    for j = 1:y1
        for k = 1:z1
            imagx(conta, k) = imag(i, j, k);
        end
        conta = conta + 1;
    end
end
% classifica
[c, u, saida, class] = Kmeans_var(imagx, 4, 500, 0.01); % os centros devem ser numeros inteiros
cria = uint8(zeros((x1 * y1), z1)); % cria área de saida unsigned int
% substitui os vetores pelos centros das classes
for i = 1:(x1 * y1)
    imagx(i, :) = c_int(class(i), :);
end

69
A.2.3 Image clustering - SOM

% cl_img_som - Classifica imagem com SOM
%
% o nome do arquivo com a imagem e 'original_corte.jpg'
%
% Ultima alteracao 24/07/2005 - RCBerredo
%
% le o arquivo e transforma de 256 x 256 x 3 para 65536 x 3
img=double(imread('original_corte.jpg')); % transforma em double
% para poder processar
[xl yl zl]=size(img); % pega dimensoes
% transforma array tri para bi-dimens. xl x yl x zl -> (xl * yl) x zl
imagx=zeros(xl.*yl,zl); conta=1; % cria area de saida
for i=1:xl
    for j=1:yl
        for k=1:zl
            imagx(i,j,k)=imag(i,j,k); % monta conjunto de treinamento
econta=conta+1;
        end
    end
end
% monta conjunto de treinamento
treina=[];
for i=1:8:(x1*y1)
    treina=[treina; imagx(i,:)]
end
% monta rede SOM e treina
net=newsom([0 255; 0 255; 0 255], [1 4], 'gridtop');
net.trainParam.epochs = 50;
net = train(net,treina');
% plota som(net.lw(1,1).net.layers(1).distances);
c_int=uint8(round(net.lw(:,1))); % pega os centros e arredonda p/ int
c=double(c_int);
% classifica
class=vec2ind(sim(net,imagx')); % monta matriz com os indices
imgy=uint8(zeros([x1.'y1],z1)); % cria area de saida unsigned int
% substitui os vetores pelos centros das classes
for i=1:(x1.'y1)
    imgy(i,:)=c_int(class(i,:));
end
% Volta ao array tri-dimensional
imagz=uint8(zeros(x1,1,y1,z1)); conta=1;
for i=1:x1
    for j=1:y1
        for k=1:z1
            imagz(i,j,k)=imagy(conta,k);
        end
        conta=conta+1;
    end
end
% salva a imagem em .bmp
imwrite(imagz,'saida_som.jpg','jpg','Quality',100);
% plota os pontos
r=imagx(:,1);
g=imagx(:,2);
b=imagx(:,3);
figure(1); plot3(r,g,b,'.'); grid on;
box on; axis tight;
title('Points Distribution','FontSize',14);
xlabel('Red','FontSize',14);
ylabel('Green','FontSize',14);
zlabel('Blue','FontSize',14);
% plota os centros
cr=c(:,1);
 cg=c(:,2);
cb=c(:,3);
figure(2); hold on; grid on;
box on; axis([0 256 0 256 0 256]);
plot3(cr,cg,cb,'r*');
plot3([0; 255],[0; 255],[0; 255],'r');
title('SOM Center Location','FontSize',14);
xlabel('Red','FontSize',14);
ylabel('Green','FontSize',14);
zlabel('Blue','FontSize',14);

A.2.4 Image clustering - SNN
% cl_img_snn - Classifica imagem com SNN
%
% Ultima alteracao 24/07/2005 - RCBerredo
%
% O nome do arquivo com a imagem e 'original_corte.jpg'

71
% Classes (saídas): 3
% Codificação: Campos receptivos fixos
% Conj. de Campos Receptivos: 1
% Tipo de Sinapses: múltiplas
%
clear all
%
% le o arquivo e transforma em double para poder processar
aux=double(imread('original_corte.jpg')); % arquivo de imagem
[x1 y1 z1]=size(aux); % pega dimensões
%
% transforma array tri para bi-dimensional x1 x y1 x z1 --> (x1 * y1) x z1
aux1=zeros((x1*y1),z1); conta=1; % aux1 = entradas analógicas
for i=1:x1
    for j=1:y1
        for k=1:z1
            aux1(conta,k)=aux(i,j,k);
        end
        conta=conta+1;
    end
end
%
% Define Parâmetros Iniciais
out_neu=4; % qtde de neurônios na saída (classes)
rho=0.1; % resolução de tempo
conj=32768; % !!!!!
% qtde de conjuntos de treinamento
tau=3; % cte de tempo do EPSP
rf=[6 6 6]; % qtde de neurônios por entrada
sigma=[1/(1.5*(rf(1)-2)) 1/(1.5*(rf(2)-2)) 1/(1.5*(rf(3)-2))];
f_cut=0.9;
maxd=20; % intervalo de codificação
%
% Define Parâmetros de Aprendizado
d=[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]; % atrasos das sub-sinapses
w_max=1; w_min=0; % pesos máximo e mínimo
max_epoc=3; % no. máximo de épocas
t_max=30; % tempo máximo de treinamento
teta=0.35; % taxa de aprendizado
beta=0.2; % cte para janela de aprendizado
nu=5.0; % vizinhança
%
% Calcula Parâmetros
kapa=1-(nu^2)/(2*log(beta/(beta+1))); % cte para janela de aprendizado
in_neu=size(rf);
ssin=size(d,2); % qtde de neurônios na entrada
theta=9.0; % qtde de sub-sinapses % valor do limiar
%
% Inicializa Peso
w=zeros(in_neu,out_neu,ssin);
for i=1:in_neu
    for j=1:out_neu
        w(i,j,:)=w_min+rand(1,ssin).*((w_max-w_min));
    end
end
%
% Inicia Treinamento
aus=kodieren_rf(aux1,rf,maxd,sigma,f_cut,rho,0,0); % codifica entrada
ctrl_1=1; % controle do loop
delta_t=zeros(1,1,ssin);

h=waitbar(ctrl_1/max_epoch,sprintf('Executando %i de %i iteracoes',... 
  ctrl_1, max_epoch));

while ctrl_1<=max_epoch
  for z=1:conj
    tr=aus(z,:);
    [neu, inst]=w2r_schiesst(tr, t_max, w, d, tau, rho, teta, 0);
    if neu==0 & size(neu,2)==1
      for i=1:in_neu
        if tr(i)==1
          w(i,neu,:)=w(i,neu,:)-eta;
        else
          delta_t(1,1,:)=tr(i)+d-inst;  % calcula delta_t
          w(i,neu,:)=w(i,neu,:)+eta.*((1+beta).*...
            exp(-1/((delta_t+2.3).^2)/(2*kapa-2)))-beta;
        end
      ndx=find(w(i,neu,:)<w_min);  % garante limites
      for u=1:size(ndx,1)
        w(i,neu,ndx(u))=w_min;
      end
      ndx=find(w(i,neu,:)>w_max);  % garante limites
      for u=1:size(ndx,1)
        w(i,neu,ndx(u))=w_max;
      end
    end
  end
  ctrl_1=ctrl_1+1;  % incrementa contador de epocas
  waitbar(ctrl_1/max_epoch,h,sprintf('Executando %i de %i iteracoes',... 
  ctrl_1, max_epoch));
  teta=teta+(0.3*teta)/max_epoch;
end

% Define as classes com a SNN treinada
[conj, in_neu]=size(aus);
class=zeros(1,conj);  % cria tabela de classes
for i=1:conj
  tr=aus(i,:);  % conjunto de teste
  [neu, inst]=w2r_schiesst(tr, t_max, w, d, tau, rho, teta, 0);  % quem disparou?
  if size(neu,2)>1  % se deu empate...
    neu=out_neu+1;  % classe = n de classes+1
  end
  class(i)=neu;  % guarda a classe
end

% calcula os centros das classes
c=zeros(out_neu+1,size(rf,2));  % cada linha e um centro
for i=1:out_neu
  a=find(class==i);
  for n=1:size(a,2)
    c(i,1)=c(i,1)+aux1(a(n),1);
    c(i,2)=c(i,2)+aux1(a(n),2);
    c(i,3)=c(i,3)+aux1(a(n),3);
  end
  c(i,:)=c(i,:)/size(a,2);
end
% a=find(class==out_neu+1);
% for n=1:size(a,2)
%  c(out_neu+1,:)=[256 0 0];  % pinta de vermelho as classificacoes erradas
\begin{verbatim}
end
end

A.3 Single synapse architecture - Section 5.6

A.3.1 First method - Time constant adaptation

% single_syn_5a - teste de classificacao com SNN
% Ultima alteracao 07/08/2005 - RCBerredo
% Testa equivalente de sinapse simples. Transforma as sinapses
% multiplas em simples. Usa o resultado de cl_spk_5a.m com 16
% neuronios de entrada e cinco de saida.
% Cada conjunto tem 50 pontos.
% Classes (saidas): 5
% Codificacao: Campos receptivos fixos
% Conj. de Campos Receptivos: 1
% Tipo de Sinapses: simples
% Atrasos: constante de tempo do EPSP
\end{verbatim}
% Prepara Entrada
clear all;
load('single_syn.mat'); % carrega vetores de pesos w
% Em single_syn.mat estão:
% aux1 = matriz 250x2 com as entradas analógicas
% aus = matriz 250x16 com as entradas analógicas codificadas
% dx = valor de tau (instantes em que o EPSP atinge o máximo)
% wx = pesos tau (valores máximos do EPSP)
% epasp = valores dos epsp resultantes para cada uma das 80 (16x5)
% sinapses multiplos durante 30ms em intervalos de 0.1ms.
% As primeiras 16 linhas referem-se ao neurônio de saída 1,
% depois as proximas 16 ao neurônio 2, etc. Desta matriz
% foram extraídos dx e wx.

% Define Parâmetros Iniciais
[in_neu out_neu]=size(dx); % qtde de entradas e saídas
c = size(aus,1); % qtde de conjuntos a classificar
rho=0.1; % resolução de tempo
teta=12;
tmax=30;

% Inicia Classificação
class=zeros(1,conj); % cria tabela de classes
for n=1:conj
    in=aus(n,:);
    ndx=find(in~=1); % acha as entradas que disparam
    tam=size(ndx,2);
    t=0; % começa em t=0
    neu=0;
    h=figure; hold on;
    while neu==0 & t<=tmax
        out=zeros(1,out_neu); % para cada saída
        for j=1:out_neu
            for i=1:tam
                if wx(ndx(i),j)==0 | dx(ndx(i),j)==0
                    sai=0;
                else
                    dt=(t-in(ndx(i)))/dx(ndx(i),j);
                    sai=wx(ndx(i),j)*dt*exp(1-dt);
                    if sai<0
                        sai=0;
                    end
                end
                out(j)=out(j)+sai;
            end
        end
        subplot(t,t+1,teta);
        plot(t,out(j),t,teta);
        if max(out)>=teta
            neu=find(out==max(out));
            inst=t;
        end
        t=t+1;
    end
    pause; close(h);
end
if size(neu,2)>1 % se deu empate...
    neu=0; % classe = 0
end
class(n)=neu; % guarda a classe
end

% Plota resultado do teste
ptos={'b' 'k' 'or' '+'b' 'xk' 'sr' 'db' 'k' '+'r' 'ob' '+'k' 'xr' 'sb' 'dk' '.r' '+'b' 'ok' '+'r' 'xb' 'sk' 'dr'};
figure; hold on;
for i=1:out_neu
    a=find(class==i);
    for n=1:size(a,2)
        plot(aux1(a(n),1),aux1(a(n),2),char(ptos(i)));
    end
end
a=find(class==0);
for n=1:size(a,2)
    plot(aux1(a(n),1),aux1(a(n),2),char(ptos(out_neu+1)));
end

A.3.2 Second method - Axonal delay adaptation

% single_syn_5b - teste de classificacao com SNN
% % Ultima alteracao 15/08/2005 - RCBarbedo
% % Teste equivalente de sinapse simples. Transforma as sinapses
% % multiplos em simples. Usa o resultado de cl_spk_5a.m com 16
% % neuronios de entrada e cinco de saida.
% % Cada conjunto tem 50 pontos.
% % Classes (saidas): 5
% % Codificacao: Campos receptivos fixos
% % Conj. de Campos Receptivos: 1
% % Tipo de Sinapses: simples
% % Atracos: atraso axonal
%
%---------------------------------------------------------------------
% Prepara Entrada
clear all;
load('single_syn.mat'); % carrega vetores de pesos w
% Em single_syn.mat estao:
% aux1 = matriz 250x2 com as entradas analógicas
% aux = matriz 250x16 com as entradas analógicas codificadas
% dx = valor de tau (instantes em que o EPSP atinge o maximo)
% cx = dx-3
% wx = pesos (valores maximos do EPSP)
% epsp = valores dos epsp resultantes para cada uma das 80 (16x5)
% sinapses multiplos durante 30ms em intervalos de 0.1ms.
% As primeiras 16 linhas referem-se ao neurônio de saida 1,
% depois as proximas 16 ao neurônio 2, etc. Desta matriz
% foram extraídos dx e wx.
%---------------------------------------------------------------------
% Define Parametros Iniciais
[in_neu out_neu]=size(dx); % qtde de entradas e saidas
conj=size(aux,1); % qtde de conjuntos a classificar
rho=0.1; % resolucao de tempo
theta=12;
tmax=30;

76
A.3.3 Using equation (5.9)
% Ultima alteracao 18/08/2005 - RCBerredo
%
% Usa 5 conjuntos com distribuicao gaussiana com media zero.
% Cada conjunto tem 50 pontos.
%
% Classes (saídas): 5
% Codificacao: Campos receptivos fixos
% Conj. de Campos Receptivos: 1
% Tipo de Sinapses: simples
% Treinamento: adapta pesos e ctes de tempo sao funcao do peso
%
% Prepara Entrada
clear all;
aux1=dlmread('cj_spk5.txt'); % dados de entrada
%
% Define Parametros Iniciais
out_neu=5; % qtde de neuronios na saida (classes)
rho=0.1; % resolucao de tempo
conj=125; % qtde de conjuntos de treinamento
%tau=3; % cte de tempo do EPSP
rf=[8 8]; % qtde de neuronios por entrada
sigma=[1/(1.5*(rf(1)-2)) 1/(1.5*(rf(2)-2))]; % largura dos campos recep
f_cut=0.9; % intervalo de codificacao
maxd=10;
%
% Define Parametros de Aprendizado
tau_max=9.8; tau_min=0.1; % ctes de tempo maxima e minima
w_max=6.98; w_min=0; % pesos maximo e minimo
max_epoch=10; % no. maximo de epochas
% tempo maximo de treinamento
eta_w=0.035; % taxa de aprendizado p/ pesos
beta=0.26; % cte para janela de aprendizado de pesos
nu=3; % vizinhanca
dx=2.8; % deslocamento de W(t) p/ esquerda (+) ou
%
% Calcula Parametros
kapa=1-(nu^2)/(2*log(beta/(beta+1)))); % cte para janela de aprendizado
in_neu=sum(rf); % qtde de neuronios na entrada
teta=12; % valor do limiar
%
% Inicializa Pesos e Ctes de Tempo
w=w_min+rand(in_neu,out_neu).*((w_max-w_min));
tau=tau_min+rand(in_neu,out_neu).*((tau_max-tau_min));
%tau=tau_max.*0.2452.*exp(-0.07922.*w/w_max)...% +0.09115.*exp(2.149.*w/w_max));
%
% inica o treinamento
tic;
aus=kodieren_rf(aux1,rf,maxd,sigma,f_cut,rho,0,0); % codifica entrada
ctrl_1=1; % controle do loop
h=waitbar(ctrl_1/max_epoch,sprintf('Executando %i de %i iteracoes',... .ctrl_1, max_epoch));
f_times=zeros(2,1);
while ctrl_1<=max_epoch
for z=1:conj
    tr=aus(z,:);
    [neu, inst]=wer_spuckt(tr,t_max,w,tau,rho,teta,0);
f_times=[f_times [neu;inst]];  
if neu==0 & size(neu,2)==1  
    for i=1:in_neu  
        if tx(i)==-1  
            w(i,neu)=w(i,neu)-eta_w;  
            tau(i,neu)=tau_max*exp(-0.07922*(w(i,neu)/w_max)+0.09115*exp(2.149*(w(i,neu)/w_max)));  
        else  
            delta_t=tr(i)-inst;  
            w(i,neu)=w(i,neu)+eta_w*((1+beta).*exp(-((delta_t+dx).^2)/(2*kapa-2))-beta);  
            tau(i,neu)=tau_max*exp(-0.07922*(w(i,neu)/w_max)+0.09115*exp(2.149*(w(i,neu)/w_max)));  
        end  
        if w(i,neu)<0  
            tau(i,neu)=tau_min;  
        end  
    end  
    if w(i,neu)>w_max  
        w(i,neu)=w_max;  
    elseif w(i,neu)<w_min  
        w(i,neu)=w_min;  
    end  
    if tau(i,neu)>tau_max  
        tau(i,neu)=tau_max;  
    elseif tau(i,neu)<tau_min  
        tau(i,neu)=tau_min;  
    end  
end  
ctrl_l=ctrl_l+1;  
waitbar(ctrl_l/max_epoch,h,sprintf('Executando %i de %i iteracoes',...
ctrl_l, max_epoch));  
teta=teta+(0.3*teta)/max_epoch;  
show_w(w);  
end  
close(h);  
toc;  
------------------------------------------  
% testa  
[conj,in_neu]=size(aus);  
class=zeros(1,conj);  
for i=1:conj  
    tr=aus(i,:);  
    [neu,inst]=wer_spuckt(tr,t_max,w,tau,rho,teta,0);  
    if size(neu,2)>1  
        se deu empate...  
        neu=0;  
    else  
        class(i)=neu;  
    end  
end  
figure; hold on;  
for i=1:out_neu  
a=find(class==i);  
for n=1:size(a,2)  
    % Plota resultado do teste  
    ppos=[.b' .k' 'or' '+''b' 'xk' 'sr' 'db'...
          .k' '+'r' 'ob' '+'k' 'xr' 'sb' 'dk'...
          .'r' '+'b' 'ok' '+'r' 'xb' 'sk' 'dr'];  
end  
end  
79
A.4 Auxiliary functions

function [neu, inst] = wer_schiesst(in, tmax, w, d, tau, rho, teta, plt)
  % Funcao [neu, inst] = wer_schiesst(in, tmax, w, d, tau, rho, teta, plt)
  % Retorna o neuronio que dispara e o instante de disparo.
  % Entrada:
  %    in = matriz linha com as entradas (firing times)
  %    tmax = maximo intervalo de calculo. se nenhuma saida dispara
  %           ate este instante, o calculo do EPSP termina
  %    w = matriz m x n x k com os pesos sinapticos, onde m sao os
  %        neuronios de entrada, n os de saida e k sub-sinapses.
  %    d = matriz linha com os atrasos de cada sub-sinapse
  %    tau = constante de tempo para calculo do EPSP
  %    rho = resolucao de tempo
  %    teta = valor do limiar
  %    plt = se e 1 plota
  %    % Saida:
  %      neu = neuronios que disparam (Atencao: pode ser mais de um!)
  %      inst = instante de disparo (se nenhuma saida dispara inst = -1)
  %    % Ultima alteracao em 19/07/2005 - RCBerredo
  %
  %--------------------------------------------------------------------------
  [ein,aus,ssin]=size(w); % ein = entradas (neuronios), aus = saidas
  ndx=find(in==1); % acha as entradas que disparam
  tam=size(ndx,2);
  if tam==0
    neu=0; inst=-1;
    return
  end
  t=0; % começa em t=0
  neu=0;
  while neu==0 & t<=tmax
    out=zeros(1,aus);
    for j=1:aus
      for i=1:tam
        dt=t-ndx(i);
        for k=1:ssin
          dtt=(dt-d(k))/tau;
          sai=w(ndx(i),j,k)*dtt*exp(1-dtt);
          if sai<0
            sai=0;
          end
          out(j)=out(j)+sai;
        end
      end
      if plt==1
plots(aus,j,t,out(j),teta);
end
end
if max(out) == teta
    neu= find(out == max(out));
    inst=t;
end
t=t+rho;
end
if neu==0 % se acabou limite de tempo
    inst=-1;
end

function plota(p1, p2, px, py, teta)
    subplot(p1,p2); hold on;
    plot(px,py,'r','(px,teta,'r'); drawnow;

function [neu, inst] = wer_spuckt(in, tmax, w, tau, rho, teta, plt)
% Funcao [neu, inst] = wer_spucktt(in, tmax, w, tau, rho, teta, plt)
% Retorna o neuronio que dispara e o instante de disparo.
% Entrada:
% in = matriz linha com as r entradas (firing times)
% tmax = maximo intervalo de calculo. se nenhuma saida dispara
% ate este instante, o calculo do EPSP termina
% w = matriz m x n com os pesos sinapticos, onde m sao os
% neuronios de entrada e n os de saida
% tau = matriz m x n com as ctes de tempo de cada sinapese
% rho = resolucao de tempo
% teta = valor do limiar
% plt = 1 - plota o EPSP
% 0 = nao plota
% Saida:
% neu = neuronios que disparam (Atencao: pode ser mais de um!)
% inst = instante de disparo (se nenhuma saida dispara inst = -1)
% Ultima alteracao em 18/08/2005 - RCErredo
%
[ein,aus]=size(w);
ndx=find(in==1);
tam=size(ndx,2);
if tam==0
    neu=0; inst=-1;
    return
end

t=0;
neu=0;
while neu==0 & t<=tmax
    out=zeros(1,aus);
    for j=1:aus
        for i=1:tam
            dt=(t-in(ndx(i)))/tau(ndx(i),j);
            sai=w(ndx(i),j)*dt*exp(1-dt);
            if sai<0
                sai=0;
            end
            end
        end
    end

out(j)=out(j)+sai;
end
if plts=1
plot(aus,j,t,out(j),teta);
end
end
def max(out)>=teta
new=find(newt=max(out));
inst=t;
end
t=t+rho;
end
if neu=-0
  % se acabou limite de tempo
  inst=-1;
end
end
function plota(pl1, pl2, px, py, teta)
  subplot(1,1,p2); hold on;
  plot(px,py,'.-',px,teta,'r.'); drawnow;
function [c,u,saida,class] = Kmeans_var(xin, nc, maxepoca, tol)

% Esta funcao tem por objetivo determinar nc centros dado um conjunto
% de entrada xin (na forma de uma variavel) conforme descrito:
% xin = [x11 x12 ... x1j;  x21 x22 ... x2j;
%        ... ... ... ...;  xi1 xi2 ... xij]
% Onde xij sao os j vetores para os quais se busca os centros, maxepoca
% e o numero maximo de iteracoes e tol e a variação maxima (diferença
% absoluta) entre duas posições consecutivas de cada nc centro.
% O método usado e o K-means. A saida tem os seguintes componentes:
% - c e uma matriz com os n centros. Cada linha representa um centro:
% c = [c11 c12 ... c1j;  c21 c22 ... c2j;
%      ... ... ... ...;  cn1 cn2 ... cnj]
% - u e a matriz de pertinencia final, onde cada coluna corresponde a um
% centro e cada linha a um dos vetores de entrada. Todas as colunas
% (centros) sao zero, exceto para o centro ao qual pertencer o vetor,
% sendo esta coluna igual a um.
% u = [u11 u12 ... u1j;  u11 u12 ... u1j;
%      ... ... ... ...;  u11 u12 ... u1j]
% - saida e a matriz com as entradas (xin) normalizadas. Se nao for usada
% a normalizacao, saida e igual a xin.
% - class e uma matriz de uma coluna, com cada linha correspondendo a uma
% linha de "xin" ou "saida", indicando o centro ao qual pertence o vetor.
% Ultima alteracao em 28/12/2002 - RCBerredo
%
%
% Inicializar Variaveis
%
% inicializa dJ com valor maior que tol
dJ=tol+1;
epoca=0;
lin=size(xin,1);
% quantidade de vetores (linhas)
col=size(xin,2);
% quantidade de colunas

% normaliza entrada
%aux=sqrt(sum(xin.^2,2));

%for n=1:lin
% input(n,:)=xin(n,:)/aux(n);
%end

input=xin;
% nao normaliza a entrada

ndx=round(rand(1,nc).*lin);
% pega nc vetores aleatorios
% No futuro, testar para ver se tem vetores iguais...
c=zeros(nc,col);
% inicializa a matriz de centros
for n=1:nc
c(n,:)=input(ndx(n,:));
end

while (dJ>tol & epoca<maxepoca) % loop principal

u=zeros(lin,nc);
% monta a matriz de pertinencia com zeros
for vetor=1:lin
for centro=1:nc
% calcula dist. entre cada vetor e os centros
dist(centro)=sum((input(vetor,:)\-c(centro,:)).^2);
end
um=find(dist==min(dist));
% acha a menor distancia e
u(vetor,um)=1;
% poe 1 na matriz de pertinencia
end

for centro=1:nc
% para cada centro:
unos=find(u(:,centro)==1);
% acha vetores da classe no
soma1=0;
% zera somas
some2=zeros(1,col);
for membro=1:size(unos,1)
% soma distancias da classe no
soma1=soma1+sum(((input(unos(membro),:))-c(centro,:)).^2);
soma2=soma2+(input(unos(membro),:));
end
dist1(centro)=soma1;
c(centro,:)=soma2(1,:)/size(unos,1);
% calcula novo centro classe no
end
epoca=epoca+1;
J(epoca)=sum(dist1);

if epoca==1
dJ=J(epoca);
else
dJ=abs(J(epoca-1)-J(epoca));
end
end

% prepara saida
saída=input;
% constrói matriz de classes e centros
for k=1:lin
class(k,1)=find(u(k,:)==0);
end

83
x=1:1:epoca;
plot(x,J);
Title(sprintf('K-Means - Variacao = %.5g',dJ),'FontSize',14);
xlabel('Epocas','FontSize',14);
ylabel('Soma das Distancias','FontSize',14);
%----------------------------------------
Appendix B

Sites on the Internet

The purpose of this appendix is to list a few links to home pages related with the present work and from where some of the material used here was obtained. These sites will certainly be of some use to those interested on this subject.

B.1 Softwares

http://www.genesis-sim.org/GENESIS/: GENESIS (short for GEneral NEural Simulation System) is a general purpose simulation platform that was developed to support the simulation of neural systems ranging from subcellular components and biochemical reactions to complex models of single neurons, simulations of large networks, and systems-level models. GENESIS has provided the basis for laboratory courses in neural simulation at Caltech, the Marine Biological Laboratory, the Crete, Trieste, Bangalore, and Obidos short courses in Computational Neuroscience, and at least 49 universities of which we are aware. Most current GENESIS applications involve realistic simulations of biological neural systems. Although the software can also model more abstract networks, other simulators are more suitable for backpropagation and similar connectionist modeling.

http://www.enorg.org/: This site of the Catacomb neural simulation software (Components And Tools for Accessible COmputer Modeling in Biology). Catacomb is a software system for building and studying models of biological systems on scales ranging from single channels to behaving animals. It incorporates graphical tools for creating model descriptions, and a library of components for computing their behavior.

http://www.neuron.yale.edu/neuron/install/install.html: NEURON is a simulation environment for developing and exercising models of neurons and networks of neurons. It is particularly well-suited to problems where cable properties of cells play an important role, possibly including extracellular potential close to the membrane, and where cell membrane properties are complex, involving many ion-specific channels, ion accumulation, and second messengers.

http://www-ra.informatik.uni-tuebingen.de/SNNS/: SNNS (Stuttgart Neural Network Simulator) is a software simulator for neural networks on Unix workstations developed at the Institute for Parallel and Distributed High Performance Systems (IPVR) at the University of Stuttgart. The goal of the SNNS project is to create an efficient and flexible simulation environment for research on and application of neural nets.

http://www.math.pitt.edu/bard/xpp/xpp.html: Home page of G. Bard Ermentrout, professor of Computational Biology at the University of Pittsburgh, containing useful mathematics packages for exploring phase spaces and dynamical systems, including XPP.
http://minduploading.org/oss/rak-ncm.html: This site contains various programs for neural circuitry modeling, mainly octave scripts and C++ programs.

http://www.cs.cmu.edu/~dst/HHsim: This is a simple educational software designed specifically for graduate or undergraduate neurophysiology courses. The user interface can be mastered in a couple of minutes and provides many ways for the student to experiment.

B.2 Other Sites


http://www.biologymad.com/NervousSystem/NervousSystem.htm: Another independent page about neurophysiology, with animated tutorials and detailed descriptions of the main components of the nerve system.

http://carbon.cudenver.edu/~mryder/martin.html: Site made by Martin Ryder, professor of the University of Colorado at Denver. Here one can find information and writings by and about leading thinkers in cognitive science, and critics and observers of the philosophy of mind.

http://www.neuroinf.de/: The Neuroinformatics Portal Pilot project, funded through a grant from the German Ministry for Science and Education (BMBF), is part of a larger effort to promote the exchange of neuroscience data, data-analysis tools, and modeling software. It is now only a basic infrastructure needed for an optimal utilization of the many available resources at a later point in time. This Portal Pilot is already on-line to allow interested users to explore the possibilities of this prototype and thus start the feedback process needed to make this Portal a community-driven resource.

http://www.siumed.edu/~dking2/ssb/neuron.htm: Site made by David King, professor of the Southern Illinois School of Medicine. This site is a complete and well organized guide to neurophysiology and contains very interesting information on this subject.
Bibliography


