Sumário

Avalanches neurais e suas medidas psicofísicas

Daniel Penalva

05/02/2014

0.1 Avalanches Neurais

Em uma rede neural, avalanche é a ativação sequencial de neuronios, ocasionada por um estímulo externo ou pela ativação de neurônios vizinhos à um neurônio em atividade.

0.1.1 Medidas Psicofísicas: atividade média e o alcançe dinâmico

Uma medida psicofísica é uma quantificação da resposta da rede neural de acordo com a variação de estímulos. A resposta da rede pode ser estudada segundo a variação da intensidade de um mesmo estímulo, esta medida leva o nome de *alcançe dinâmico*.

A estatística apropriada para o cálculo do alcançe dinâmico é a *atividade média* da rede, que para neurônios discretos toma a forma de

$$F_T(r) = \sum_{t=0}^{T} \frac{\rho_t(r)}{T}, \tag{1}$$

ρ_t é a densidade instantânea de neurônios ativos. A estatística $F_T(r)$ é tomada para longos períodos de tempo (1000τ), quando é assumida como estacionária, e tem o comportamento cumulativo, pois o aumento na intensidade do estímulo r causa saturação na atividade da rede.

O alcançe dinâmico é então definido como a variação, em r, necessária para que a atividade média varie 80%, é usualmente expressa em decibeis

$$\Delta = 10 \log \frac{r_{90}}{r_{10}} \tag{2}$$

0.1.2 Capacidade e eficiência de informação de uma rede

Avalanches neurais nos permitem elaborar perguntas pertinentes ao uso prático de redes neurais, que presumidamente fazem cálculos computacionais, quais estão relacionados à informatica das redes neurais. O tamanho das avalanches, definido como o número total de neurônios ativados s durante uma atividade, diz a respeito da *capacidade de informação* das redes, a duração das avalanches nos levará à *eficiência de informação das redes*.

O acesso à informática das redes é dado pelos histogramas de tamanho de avalanche p_s, de duração de avalanche p_t e pela entropia de Shannon

$$H(\{p_n\}) = -\sum_n p_n \log p_n \tag{3}$$

0.1.3 Otimização crítica simultânea do alcançe dinâmico e da eficiência informática

Evidências de maximização do alcançe dinâmico, em redes com regimes críticos !TODO Explicar na tese o que sao fenomenos criticos!, foram encontradas no

exame do automata de Kinouchi-Copelli [1]. Posteriormente Mosqueiro e Maia [4] mostraram, para mais de um tipo de topologia, que a informática relevante para otimização do alcançe dinâmico não é a capacidade da rede e sim a eficiência da rede. A eficiência de informação é sempre maximizada junto com o alcançe dinâmico, ao contrário a capacidade de informação pode crescer independentemente do alcançe dinâmico. Mosqueiro e Maia ainda encontraram evidências de que essa propriedade pode ser explicado por dois fatores: tipo de conectividade da rede e período de refratoriedade de cada neurônio discreto do tipo Kinouchi-Copelli.

Em trabalhos anteriores Larremore et.al. mostraram equações analíticas relacionando o alcançe dinâmico e propriedades espectrais da matrix de adjacência [3, 2], cujos pesos são dados pelas probabilidades de transmissão de atividade, bem como com o período de refratário, tanto para o modelo de Kinouchi-Copelli quanto para uma generalização do modelo, que inclui delay temporal diverso na transmissão do sinal e multiplos períodos de refratoriedade. É esperado que esses resultados possibilitem uma análise analítica da informática de redes neurais, para o caso do automata de Kinouchi-Copelli, devido às evidências encontradas por Mosqueiro e Maia ligando o alcançe dinâmico às informáticas descritas.

Redes cerebrais complexas Daniel Penalva 20/02/2014

0.2 Caracterizações comuns do cérebro

0.2.1 Redes Anatômicas

Os dados revelam que o cérebro apresenta evidências de redes de pequeno mundo, tanto estrutural como funcional.

0.2.2 Redes Funcionais

Código e pseudo-código de automatos cíclicos em redes complexas e multiplexas Daniel Penalva 24/01/2014

0.3 O Automato de Kinouchi-Copelli

O termo automato designa uma unidade dinâmica determinística ou estocástica (aleatória) caracterizada por espaços de estados enumerados em tempo discreto. Geralmente o termo é utilizado para se referir á sistemas com multiplas unidades com regras de interações locais que guiam a dinâmica. O estado do i-ésimo automato de Kinouchi Copelli toma um valor natural s_i por vez, caracterizado pelo número finito de estados possýeis $s_i = 0, 1, ..., m-1$, ele é cíclico e possui regras de interações locais e estocásticas.

Introduzido para o estudo do alcançe dinâmico ótimo de uma rede neural por Kinouchi e Copelli [1], ele é caracterizado pelas seguintes regras de transição:

1. $s_i = 0 \rightarrow 1$, transição do estado quiescente para o de atividade, com probabilidade $P_{s_i}(0 \rightarrow 1) = 1 - e^{-r\tau}\Pi_{j=0}^{k_i}[1 - p_{ij}]$,

2. todas as demais transições são deterministas,

onde r é a taxa de entrada para um processo de poisson externo e independente das transmissões internas à rede e τ é a unidade de tempo considerada para a rede (normalmente 1 ms).

0.3.1 Modelo em redes complexas e pseudo-código

Algorithm 1 Automato de Kinouchi-Copelli em redes complexas

```
1: Função KC COMPLEXAS( estado inicial dos indivíduos, sensores, rede, tempo total)
2:     estado atual dos indivíduos ← estado inicial dos indivíduos
3:     Para instante em tempo total faça:
4:         estado anterior dos indivíduos ← estado atual dos indivíduos
5:         Para indivíduo na rede faça:
6:             Se estado anterior do indivíduo == 0 então:
7:                 probabilidade de transição ← PRIMEIROS VIZINHOS ATIVOS( indivíduo, rede, sensores)
```

Repare que a probabilidade de transição, retornada pela função *Primeiros vizinhos ativos*, que se baseia na rede e no nó em questão, é pré-afixada antes do algoritmo *KC complexas* ser executado, a função *Primeiros vizinhos ativos* apenas seleciona a probabilidade de transição correta.

No artigo original, as probabilidades pré-afixadas de transmissão entre indivíduos são sorteados de uma distribuição uniforme $[0, p_{max}]$, $p_{max} = 2\frac{\sigma}{<k>} < 1.0$.

É importante notar que o argumento *sensores* determina se o indivíduo recebe sinal externo, sua implementação pode estar implícita no argumento rede. Ele altera a probabilidade de transição, no paper original o sinal de entrada é um processo de poisson, de taxa **r**, independente das probabilidades pré-afixadas para transmissão de sinal entre nós.

Repare que é necessário implementar, junto com o algoritmo da simulação, mecanismos de gravação dos estados da rede, para que seja possível realizar medidas estatísticas descritas na introdução.

8: **Se** (número sorteado entre 0 e 1) < probabilidade de transição **então:**
9: estado atual do indivíduo ← estado atual do indivíduo + 1
10: **final de Se**
11: **ou então:**
12: estado atual do indivíduo ← (estado atual do indivíduo +1)%m
13: **final de Se**
14: **final de Para**
15: **final de Para**
16: **final de Função**

0.3.2 Modelo em redes multiplexas e pseudo-código

0.4 Código em Python

0.4.1 Sobre a codificação de um algoritmo

O código abaixo implementa o pseudo-código discutido acima. Como se trata de uma redação literária livre do pseudo-código (uso de estruturas de dados, fluxos de dados e nomenclaturas arbitrárias, baseando-se no pseudo-código) de punho do autor deste documento, é de escolha deste autor que o código seja publicado com licença AGPL.

0.4.2 Código: Multi topologies network and cyclic Kinouchi-Copelli like automatas

```
1  import networkx as nx
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from matplotlib import rc
5
6  def kc( ativos0, num_estados, rede, tempo):
7      firstN = set();
8      ativos = set();
9      ativos.add(ativos0);
10     firstN.add(ativos0);
11     deterministicos = set();
12     N = len(rede);
13     for instante in tempo:
14 #         print " Tempo "+str(instante)+"\n";
15 #         print " Ativos"+str(ativos)+"\n";
16         deterministicos.update(ativos);
17 #         print " Transicao deterministica "+str(
       deterministicos)+"\n";
18         ptrans = primeirosVizinhosAtivos( ativos, rede);
19 #         print " Rede "+str(rede.edge)+"\n";
20         lt = len(ptrans);
21         transicoes = np.random.rand(lt);
22         c = 0;
23 #         print " sorteados "+str(transicoes)+"\n"+" 1-p "+
       str(ptrans)+"\n";
24         for t in ptrans:
25             if transicoes[c] < (1.0-ptrans[t]):
26                 rede[t]['estado'] = rede[t]['estado'] + 1;
27                 ativos.add(t);
28                 c = c + 1;
29             else:
30                 c = c + 1;
31         for b in deterministicos.copy():
```

```
32              rede[b]['estado'] = np.mod( rede[b]['estado'] +
                    1, num_estados);
33              if rede[b]['estado'] == 0:
34                  deterministicos.remove(b);
35              if rede[b]['estado'] != 1:
36                  ativos.remove(b);
37  #          print " Rede "+str(rede.edge)+"\n";
38
39          Num = len(ativos);
40          if Num == 0 or Num > int(0.9*N):
41              break;
42          firstN.update(ativos);
43      return (len(firstN),instante);
44
45  ## Selecao de probabilidade, primeira tentativa
    produtorio_ativos(1-p). probabilidade de nao chegar
    nenhum sinal aos vizinhos, a probabilidade de chegar pelo
     menos um sinal eh 1-p_vizinhos ##
46  def primeirosVizinhosAtivos(ativos, net):
47      p_vizinhos = {};
48      for i in ativos:
49          for j in net[i]:
50              if j in p_vizinhos:
51                  p_vizinhos[j] = p_vizinhos[j]*(1.0 - net[i][
                        j]['P']);
52              else:
53                  if j != 'estado' and j != 'input':
54                      if net[j]['estado'] == 0:
55                          p_vizinhos[j] = net[i]['input']*(1-
                                net[i][j]['P']);
56  #     for i in p_vizinhos:
57  #         l = len(p_vizinhos[i]);
58  #         prod = 1.0;
59  #         print "\n Neighb_num "+str(p_vizinhos)+" \n"
60  #         for j in range(l):
61  #             prod = prod*(1.0-p_vizinhos[i][j]);
62  #         p_vizinhos[i] = 1 - net[i]['input']*prod;
63  #     print "\n Quiescent transition P "+str(p_vizinhos)+" \n
    ";
64      return p_vizinhos;
65
66  def shrink(a):
67      a = list(a);
68      a.sort();
69      for el in a:
70          while a.count(el) > 1:
71              a.remove(el);
72      return np.array(a);
73
74  def allIndex(rede, a, diverse = False):
75      N = len(rede);
76      indexes = [];
77      if diverse:
78          for i in range(N):
79              if rede[i]['estado'] != a:
80                  indexes.append(i);
81          return indexes;
82      else:
83          for i in range(N):
84              if rede[i]['estado'] == a:
85                  indexes.append(i);
86          return indexes;
87
88  def associarEstadosRede(rede, estados):
89      for i in range(len(rede)):
```

```
90          rede[i]['estado'] = estados[i];
91
92  def associarEstimuloExterno(rede, individuos, taxa):
93      for i in range(len(rede)):
94          rede[i]['input'] = 1.0;
95      for member in individuos:
96          rede[individuos.pop()]['input'] = np.exp(-taxa);
97
98  def associarProbabilidades(rede0, pmax):
99      for edge in rede0.edge:
100         for a in rede0.edge[ edge ].items():
101             if a[0] == (edge-1):
102                 rede0.edge[ edge ][ a[0] ][ 'P' ] = rede0.
                        edge[ edge -1][ edge ][ 'P' ];
103             else:
104                 rede0.edge[ edge ][ a[0] ][ 'P' ] = np.
                        random.uniform(0, pmax);
105
106 def histogram(data,bins, minimum = -7.5, maximum = 7.5, plot
        = False):
107     binsize = (maximum - minimum)/float(bins);
108     binxpos = [minimum + binsize*j for j in np.arange(bins)
            ];
109     binypos = [];
110     data.sort();
111     for i in range(bins):
112         s = data.searchsorted([minimum + i*binsize,minimum +
                (i+1)*binsize]);
113         binguys = data[s[0]:s[1]];
114         binxpos.append(binguys.size);
115         norm = float(binsize*data.size) #float(sum(binypos))
                ;
116         binypos = [j/norm for j in binypos];
117     if plot == True:
118         plt.bar(binxpos,binypos,binsize,color='red');
119     else:
120         return (binxpos,binypos);
121
122
123 def simulateSizeTime(num_est = 2, Neurons = 2500, km = 10.0,
        sigma = .85, tempom = 10000, realiza = 500, inptax =
        1.0, inpset= set()):
124     numero_estados = num_est;
125     N = Neurons;
126     k_mean = km;
127 # Calcula <k> = np.mean(rede0.degree(rede0).values());
128     tempot = np.linspace(0, tempom-1, tempom);
129     pmax = 2.*sigma/k_mean;
130     e0 = np.zeros(N);
131     estimulo = inptax;
132 ## Condicao inicial aleatoria
133 # e0 = np.random.randint(0, numero_estados, N);
134 ## Para o caso de haver individuo com entradas externas do
        tipo processo de Poisson
135 #inp = np.random.randint(0, N, N/10);
136 #inp = shrink(inp);
137     S_ize = [];
138     T_ime = [];
139     for i in range(realiza):
140         rede0 = nx.generators.gnm_random_graph(N, N*k_mean
                /2);
141         associarProbabilidades(rede0, pmax);
142         associarEstadosRede(rede0, e0);
143         associarEstimuloExterno(rede0, inpset, inptax);
144         inicial = np.random.randint(0,N);
```

```
145         rede0[inicial]['estado'] = 1.0;
146         (a,b) = kc(inicial ,numero_estados, rede0, tempot);
147         S_ize.append(a);
148         T_ime.append(b);
149         print "\n";
150         print i;
151 
152     S_ize.sort()
153     T_ime.sort()
154 
155     l1 = S_ize.count(1);
156     l2 = T_ime.count(0);
157 
158     for i in range(l1):
159         S_ize.pop(0);
160     for i in range(l2):
161         T_ime.pop(0);
162 
163     return (S_ize,T_ime);
164 
165 def networkScaling(num_est = 2, neurons_min = 1000,
    neurons_max = 10000, neurons_step = 250,km = 10.0, tempom
    = 10000):
166     numero_estados = num_est;
167     k_mean = km;
168 # Calcula <k> = np.mean(rede0.degree(rede0).values());
169     tempot = np.linspace(0, tempom-1, tempom);
170 ## Condicao inicial aleatoria
171 # e0 = np.random.randint(0, numero_estados, N);
172 ## Para o caso de haver individuo com entradas externas do
    tipo processo de Poisson
173 #inp = np.random.randint(0, N, N/10);
174 #inp = shrink(inp);
175     S_igma = [];
176     counter = 0;
177     for N in range( neurons_min, neurons_max+1, neurons_step
        ):
178         a = N;
179         print " N "+str(N);
180         sigma = 1.5;
181         counter = 500;
182         while counter > 0:
183             print "sigma "+str(sigma)
184             pmax = 2.*sigma/k_mean;
185             rede0 = nx.generators.gnm_random_graph(N, N*
                k_mean/2);
186             associarProbabilidades(rede0, pmax);
187             e0 = np.zeros(N);
188             associarEstadosRede(rede0, e0);
189             associarEstimuloExterno(rede0, set(), 0.0);
190             inicial = np.random.randint(0,N);
191             rede0[inicial]['estado'] = 1.0;
192             (a,b) = kc(inicial ,numero_estados, rede0,
                tempot);
193             if a > 0.9*N:
194                 sigma = sigma - 0.1;
195             else:
196                 counter = counter - 1;
197         S_igma.append(sigma)
198     return (S_igma, np.linspace(neurons_min,neurons_max+1,
        len(S_igma)))
199 
200 (s,t) = networkScaling();
```

Referências Bibliográficas

[1] Osame Kinouchi and Mauro Copelli. Optimal dynamical range of excitable networks at criticality. *Nature Physics*, 2(5):348–351, 2006.

[2] Daniel B Larremore, Woodrow L Shew, Edward Ott, and Juan G Restrepo. Effects of network topology, transmission delays, and refractoriness on the response of coupled excitable systems to a stochastic stimulus. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 21(2):025117–025117, 2011.

[3] Daniel B Larremore, Woodrow L Shew, and Juan G Restrepo. Predicting criticality and dynamic range in complex networks: effects of topology. *Physical review letters*, 106(5):058101, 2011.

[4] Thiago S Mosqueiro and Leonardo P Maia. Optimal channel efficiency in a sensory network. *arXiv preprint arXiv:1204.0751*, 2012.