UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE CIÊNCIAS E TECNOLOGIA (CCT)
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA
DOUTORADO EM ENGENHARIA ELÉTRICA

ROSÂNGELA MARQUES DE ALBUQUERQUE

REDES DE PETRI COLORIDAS PARA REPRESENTAÇÃO DO CONHECIMENTO E APRENDIZADO DE REDES NEURAIS ARTIFICIAIS

FORTALEZA
2023

ROSÂNGELA MARQUES DE ALBUQUERQUE

REDES DE PETRI COLORIDAS PARA REPRESENTAÇÃO DO CONHECIMENTO E APRENDIZADO DE REDES NEURAIS ARTIFICIAIS

Tese apresentada ao Programa de Pós-Graduação em Engenharia Elétrica do Centro de Ciências e Tecnologia (CCT) da Universidade Federal do Ceará, como requisito parcial à obtenção do título de doutor em Engenharia Elétrica. Área de Concentração: SISTEMAS DE ENERGIA ELÉTRICA

Orientador: Prof. Dr. Giovanni Cordeiro Barroso

Coorientador: Prof. Dr. Guilherme de Alencar Barreto

FORTALEZA

2023

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

A313r Albuquerque, Rosângela Marques de.
Redes de Petri Coloridas Para Representação do Conhecimento e Aprendizado de Redes Neurais Artificiais / Rosângela Marques de Albuquerque. – 2023.
76 f. : il. color.

Tese (doutorado) – Universidade Federal do Ceará, Centro de Tecnologia, Programa de Pós-Graduação em Engenharia Elétrica, Fortaleza, 2023.
Orientação: Prof. Dr. Giovanni Cordeiro Barroso.
Coorientação: Prof. Dr. Guilherme de Alencar Barreto.

1. Redes Neurais Artificiais. 2. Perceptron Multicamadas. 3. Redes de Petri coloridas. 4. Backpropagation. 5. Modelo de MCCulloch-Pitts. I. Título.

CDD 621.3

ROSÂNGELA MARQUES DE ALBUQUERQUE

REDES DE PETRI COLORIDAS PARA REPRESENTAÇÃO DO CONHECIMENTO E APRENDIZADO DE REDES NEURAIS ARTIFICIAIS

Tese apresentada ao Programa de Pós-Graduação em Engenharia Elétrica do Centro de Ciências e Tecnologia (CCT) da Universidade Federal do Ceará, como requisito parcial à obtenção do título de doutor em Engenharia Elétrica. Área de Concentração: SISTEMAS DE ENERGIA ELÉTRICA

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. Giovanni Cordeiro Barroso (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Guilherme de Alencar Barreto (Coorientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Francisco Milton Mendes Neto
Universidade Federal Rural do Semi-Árido (UFERSA)

Prof. Dr. Corneli Gomes Furtado Júnior
Instituto Federal do Ceará (IFCE)

Prof. Dra. Ruth Pastôra Saraiva Leão
Universidade Federal do Ceará (UFC)

À minha filha, pelo seu grande amor por mim... Pelo esforço que você fazia em manter-se acordada até eu chegar em casa, só para me abraçar, me beijar e dizer que estava com saudades. Por compreender as vezes que não pude ir a algum evento da escola. Por estar ao meu lado com muito amor e carinho, mesmo quando o mundo parecia desabar, sua presença me motivou e me deu a certeza de que não estou sozinha nessa caminhada.

AGRADECIMENTOS

Antes de tudo, quero agradecer a Deus, por ter abençoado todos os dias da minha vida, por iluminar meu caminho e me dar forças para seguir sempre em frente. Quero agradecer também as pessoas que fizeram parte da minha trajetória.

À minha mãe e minhas irmãs, pelo incentivo aos estudos desde minhas primeiras séries escolares.

Ao Janes, meu namorado, pelo permanente incentivo e por ter me ajudado com as figuras deste meu trabalho. Agradeço ainda a paciência e amor demonstrados nos meus momentos difíceis.

Ao Cléssio, por sempre cuidar da minha filha enquanto eu viajava. Obrigada pelo amor e carinho que você cuidou dela.

Aos meus queridos professores do mestrado, em especial ao Vandilberto, Arthur e Jarbas, cujos conselhos, aulas e motivação me inspiraram para realizar a pesquisa desta tese.

Aos queridos funcionários que tornaram nosso dia-a-dia não somente possível, como agradável. Obrigada pelo trabalho valioso que vocês prestam todos os dias com limpeza, conservação e manutenção do espaço acadêmico e principalmente por fazerem esse trabalho com alegria e simpatia. Senti muitas saudades de vocês, quando foram convidados a se retirar do departamento no horário do almoço.

Aos colegas de curso, em especial ao Darielson, Josias, Fellipe, Kaio, Lucas e Vanessa, cujas conversas e brincadeiras tornaram o curso mais leve e divertido. Obrigada pela amizade e por serem amigos com quem pude contar sempre.

Aos meus coorientadores, Cornelli e Guilherme. Se consegui terminar esta tese, “foi porque subi em ombros de gigantes”.

Ao meu querido orientador Giovanni, tenho muito orgulho de tê-lo como orientador, admiro sua dedicação e amor ao trabalho. Agradeço pela confiança, pela amizade, conselhos e paciência. O senhor é um exemplo de simplicidade, compreensão e competência. Nas reuniões semanais, aprendi a trabalhar em grupo e principalmente, a me colocar no lugar do outro. Minhas idas a sua sala era um lugar de apoio e palavras de sabedoria que acalmava meu coração aflito. Muitíssimo Obrigada! E que eu possa sempre contar com o privilégio da sua amizade.

"O sonho é que leva a gente para frente. Se a gente for seguir a razão, fica aquietado, acomodado."

(Ariano Suassuna)

RESUMO

As *Redes Neurais Artificiais* (RNAs) compreendem ferramentas computacionais paralelas e distribuídas que podem aprender com dados e fazer inferências (ou seja, previsões) para sistemas não lineares. Por sua vez, as *redes de Petri* (RP) consistem em ferramentas de modelagem bem estabelecidas para sistemas a eventos discretos e distribuídos com uma série de contribuições bem-sucedidas para automação e controle de tarefas industriais complexas. Assim, motivados por um interesse em usar o formalismo de PNs para emular ou projetar arquiteturas de redes neurais, revisitamos este tópico de pesquisa recorrendo à poderosa estrutura de modelagem de PNs coloridas temporizadas hierárquicas (*hierarchical timed colored PNs*, HTCPNs) para introduzir uma nova abordagem que constrói um modelo perceptron multicamadas (*multilayer perceptron* (MLP)) totalmente adaptável de uma camada oculta treinado pelo algoritmo de retropropagação. O modelo proposto resultante é denominado HTCPN-MLP e consiste em uma estrutura geral capaz de lidar com tarefas de classificação e regressão. Para desenvolver o modelo HTCPN-MLP, um modelo PN colorido do tipo perceptron (*Perceptron-CPN* (P-CPN)) é primeiro construído sobre um novo modelo PN colorido de McCulloch-Pitts (*McCulloch-Pitts-CPN* (MP-CPN)) de um neurônio, sendo esta outra contribuição do trabalho atual. Um conjunto de experimentos é apresentado a fim de destacar a capacidade de aprendizado do modelo HTPCN-MLP proposto e suas vantagens em relação aos modelos alternativos disponíveis na literatura.

Palavras-chave: redes neurais artificiais; perceptron multicamadas; perceptron; modelo de MCCulloch-Pitts; redes de Petri coloridas; backpropagation.

ABSTRACT

Artificial neural networks (ANNs) comprise parallel and distributed computational tools that can learn from data and make inferences (i.e., predictions) for highly nonlinear systems. By its turn, Petri nets (PNs) consist of well established modeling tools for parallel and distributed discrete event systems with a number of successful contributions to automation and control of complex industrial tasks. Thus, motivated by a long lasting interest in using the formalism of PNs either to emulate or to design neural network architectures, we revisit this research topic by resorting to the powerful modeling framework of hierarchical timed colored PNs (HTCPNs) to introduce a novel approach that builds a fully adaptive one-hidden-layered multilayer perceptron (MLP) model trained by the famed backpropagation algorithm. The resulting proposed model is called HTCPN-MLP and consists of a general structure capable of handling classification and regression tasks. In order to develop the HTCPN-MLP model, a perceptron-like colored PN (perceptron-CPN) model is first built upon a novel McCulloch-Pitts colored PN (McCulloch-Pitts-CPN) model of a neuron, this being another contribution of the current work. A pedagogical set of experiments is presented in order to highlight the learning capability of the proposed HTPCN-MLP model and its advantages with respect to alternative models available in the literature.

Keywords: multilayer perceptron; colored Petri nets; backpropagation; learning; formal model.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ANNs	*Artificial neural networks*
ANP	*Adaptive Neural Processor*
CA-CPN	*constructive algorithms for colored Petri net*
CTNPN	*redes de Petri neurais temporizadas coloridas*
DES	*sistemas a eventos discretos*
HOPN	*Higher-Order Petri Net*
HTCPN-MLP	*Rede de Petri hierárquica colorida Temporizada Perceptron Multicamadas*
HTCPNs	*PNs coloridas temporizadas e hierárquicas*
MLP	*multilayer perceptron*
MLP-MAT	*MLP implementada no Matlab*
MP	*McCulloch-Pitts*
MP-CPN	*McCulloch-Pitts-CPN*
MSE	*erro quadrático médio*
NTP	*Neuron Type Processor Modeling Using a Time Petri Net*
P-CPN	*Perceptron-CPN*
RNAs	*Redes Neurais Artificiais*
RP	*redes de Petri*
RPC	*redes de Petri coloridas*
WFRN	*Wall-Following Robot Navigation*

SUMÁRIO

1 INTRODUÇÃO

Os avanços nas RNAs são motivados por uma ampla gama de aplicações modernas no mundo real, que incluem classificação de padrões, agrupamento, regressão não linear, otimização e problemas de controle. Na classificação de padrões, as RNAs podem detectar padrões em um conjunto de dados por meio de aprendizado supervisionado (ARIVUDAINAMBI *et al.*, 2019). Outra aplicação é o aprendizado não supervisionado, no qual a RNA atribui dados semelhantes a um determinado cluster (ROWSHANDEL *et al.*, 2018). Em aplicações que envolvem aproximação de funções, as RNAs podem ser aplicadas a problemas em que um modelo analítico não está disponível, ou seja, aproximam a relação entrada-saída de um determinado problema de interesse a partir de um determinado conjunto de dados (ROBLES-ALGARÍN *et al.*, 2019). Além disso, uma RNA pode ser treinada com dados sequenciais para aplicação em previsão de séries temporais ou identificação de sistemas (CABANEROS *et al.*, 2019). Também é possível lidar com problemas de otimização encontrando soluções que maximizem ou minimizem funções sujeitas a diferentes restrições (SI *et al.*, 2019). Por último, mas não menos importante, também é possível determinar as entradas que permitirão obter o comportamento desejado do sistema em aplicações de controle (BOUHOUNE *et al.*, 2017).

As RNAs possuem uma estrutura organizacional inspirada na estrutura do cérebro humano, contendo várias camadas de neurônios que processam informações de forma paralela e distribuída. As habilidades de aprendizado e processamento de informações não lineares estão entre as principais razões por trás das aplicações bem-sucedidas de RNAs a problemas do mundo real. No entanto, aplicações para *sistemas a eventos discretos* (DES) são um tanto limitadas e escassas na literatura. Estas são mantidas principalmente pelas Redes de Petri (MURATA, 1989), que são um paradigma gráfico para a descrição formal das interações lógicas entre partes ou do fluxo de atividades em sistemas distribuídos complexos. Em outras palavras, as redes de Petri (*Petri Net*, PNs) são usadas para representar a estrutura causal da operação concorrente e cooperação dos componentes de um sistema distribuído (LI *et al.*, 2022).

Apesar dessa dominância de PNs para lidar com processamento de informações distribuídas em DES, observa-se há muito tempo um interesse crescente em usar o formalismo de modelagem abrangente de PNs para desenvolver modelos de aprendizado de máquina para representação de conhecimento, tomada de decisão e aprendizagem. Uma primeira tentativa nesse sentido é a rede de Petri Lugar/Transição proposta (GENRICH; THIELER-MEVISSEN, 1976) para representar o conhecimento por meio do cálculo de predicados de primeira ordem. O

interesse na representação do conhecimento baseado em regras por meio de PNs continuou em trabalhos posteriores, mudando para uma abordagem mais orientada à lógica fuzzy (CHAN *et al.*, 1990; GARG *et al.*, 1991; LIU *et al.*, 2017; LIU *et al.*, 2021; XU *et al.*, 2019), devido à sua capacidade de lidar com incerteza e ambiguidade.

Também houve tentativas de desenvolver modelos PN para o neurônio artificial. Partindo do modelo de neurônio padrão *McCulloch-Pitts* (MP), as PNs são usadas como um bloco de construção de redes neurais simples (AHSON, 1995) e para implementar circuitos lógicos digitais para redes neurais (HABIB; NEWCOMB, 1990a). A PN de ordem superior foi introduzida em (CHOW; LI, 1997) para representar o modelo do neurônio MP e as redes neurais de ordem superior usando o formalismo das redes de Petri.

De particular interesse para o trabalho atual são os modelos baseados em PN desenvolvidos em (KORIEM, 2001a) e (SHEN *et al.*, 2010). Em (KORIEM, 2001a), é proposto o conceito de *redes de Petri neurais temporizadas coloridas* (CTNPN) (ou shortly CN-net) que são isomórficas a arquiteturas neurais. A técnica CN-net incorpora os recursos básicos de representação de RNAs feedforward usando os recursos de modelagem de redes de Petri coloridas e temporizadas. No entanto, a CN-net proposta não aborda a capacidade de aprendizagem inerente das RNAs como uma característica, mas sim como algo que tem potencial para ser desenvolvido no futuro, e não há desenvolvimentos a esse respeito. Seguindo uma abordagem diferente, os autores em (SHEN *et al.*, 2010) desenvolveram uma PN que simula uma rede neural, não uma RNA que é construída usando o formalismo PN. O modelo de PN resultante é de fato adaptativo (ou seja, treinável), mas não emprega um algoritmo de aprendizado no sentido de rede neural, mas de qualquer forma fornece evidências do grande potencial de PNs como ferramenta de modelagem para simular e projetar redes neurais.

A partir do exposto, vários autores atestaram que as PNs são de fato fortes ferramentas de modelagem para a construção de arquiteturas de RNAs multicamadas totalmente treináveis, devido à sua capacidade de representar modelos gráficos para sistemas de processamento de informações distribuídas. No entanto, esses modelos anteriores não foram capazes de fornecer até o momento uma caracterização completa das habilidades de aprendizado das arquiteturas de RNA feedforward, incluindo uma implementação escalável do algoritmo backpropagation (BP).

Nesse sentido, o presente trabalho visa desenvolver a partir do zero um modelo do tipo MLP que possa ser totalmente treinada pelo algoritmo BP usando o formalismo de modelagem de *PNs coloridas temporizadas e hierárquicas* (HTCPNs). O modelo MLP baseado

em PN proposto, denominado doravante HTCPN multi-layer perceptron (HTCPN-MLP), é o produto final de uma série de desenvolvimentos menores, mas não menos importantes que inicia com uma implementação baseada em CPN do modelo de neurônios McCulloch-Pitts. Muito mais do que implementações diretas de arquiteturas de RNA, os modelos propostos são, na verdade, sofisticados modelos adaptativos baseados em PN. Tendo isso em mente, o modelo HTCPN-MLP proposto pode certamente ampliar a gama de aplicações de PNs para tarefas em que as RNAs claramente superam modelos não adaptativos padrão, como detecção e classificação de falhas.

Até onde sabemos, é a primeira realização de uma rede MLP totalmente treinável usando o formalismo lógico-matemático de PNs coloridas temporizadas e hierárquicas. Por fim, essa contribuição também ajuda a abrir caminho para o desenvolvimento de modelos de aprendizado de máquina mais complexos usando as habilidades de representação de conhecimento dos HTCPNs.

1.1 Objetivo Principal e Objetivos Específicos

O objetivo principal deste trabalho é fornecer um modelo completo baseado em PN de uma rede neural MLP com uma implementação totalmente funcional do algoritmo de aprendizado de retropropagação do erro (RUMELHART *et al.*, 1986). Os objetivos específicos estão citados abaixo de acordo com a ordem de evolução:

- Construção de um modelo de neurônio em uma rede de Petri colorida;
- Construção de um modelo de rede percetron simples em uma rede de Petri colorida;
- Construção de um modelo de uma rede neural artificial MLP em uma rede de Petri colorida, temporizada e hierárquica que possa aprender com a experiência e fazer inferências;
- Construção de um modelo de uma rede neural construtiva em uma rede de Petri colorida, temporizada e hierárquica.

1.2 Organização do Trabalho

O presente trabalho está organizado na seguinte forma: No Capítulo 2 são apresentadas as redes de Petri, as principais definições e suas propriedades. A seguir são apresentadas as redes de Petri de alto nível: as redes de Petri coloridas, temporizadas e hierarquicas, mostrando suas definições e o alto poder de modelagem que elas possuem. Nas seções seguintes deste

capítulo é abordado o modelo de McCulloch-Pitts que é o modelo matemático para o neurônio biológico, a rede neural perceptron e a perceptron multicamadas.

No Capítulo 3 são desenvolvidos e implementados: uma rede de Petri colorida para o modelos de McCulloch-Pitts-CPN. Estendemos o modelo do neurônio e desenvolvemos um modelo de rede de Petri colorida perceptron (perceptron-CPN), aplicado a dois problemas de classificação linearmente separável como prova de conceito.

No Capítulo 4 é desenvolvida e implementada uma rede de Petri colorida temporizada e hierarquica para a rede neural perceptron multicamada construtiva, inspirada no processo biológico que produz o cérebro humano, inicia-se com uma estrutura mínima, novos neurônios são adicionados um a um a medida que necessários, viabilizando a construção dinâmica da arquitetura da rede neural, enquanto realiza o seu treinamento. O *Rede de Petri hierárquica colorida Temporizada Perceptron Multicamadas* (HTCPN-MLP) com o algoritmo de aprendizado backpropagation é formalmente apresentado e desenvolvido passo a passo.

No Capítulo 5 realizamos a análise de desempenho do HTCPN-MLP em alguns bancos de dados amplamente conhecidos: íris, dermatologia, vinho, navegação por robôs de seguimento de parede, cardiotocografia e colisões sísmicas (iris, dermatology, wine, Wall-Following Robot Navigation, Cardiotocography and seismic-bumps).

No Capítulo 6 propomos uma rede de Petri colorida para análise e modelagem das redes neurais construtivas, a rede se constroi à medida que vai fazendo o seu treinamento.

No Capítulo 7 são apresentados as conclusões e trabalhos futuros desta tese.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo é apresentada uma revisão bibliográfica sobre os tópicos envolvidos neste trabalho, inicia-se com as redes de Petri: fundamentação teórica, formalismo, representação gráfica, poder de modelagem, redes de Petri de alto nível: coloridas, temporizadas e hierarquicas. Depois, redes neurais artificiais: história e topologia, por último, uma comparação do modelo proposto nesta tese com os modelos da literatura.

2.1 Redes de Petri

As redes de Petri foram originalmente descritas por Carl Adam Petri em sua tese de doutorado intitulada Comunicação entre Autômatos, defendida em 1962 (PETRI, 1962). A partir de então, consideráveis trabalhos teóricos e aplicações práticas com PNs têm sido realizados, principalmente nas áreas de modelagem, redes de computadores, comunicações, sistemas distribuídos, sistemas elétricos de potênica, protocolos de comunicação, sistemas operacionais, sistemas de controle de produção, automação industrial, modelagem de controladores lógicos, análise de fluxo de tarefas, chips VLSI, modelagem de sistemas a eventos discretos (JENSEN; KRISTENSEN, 2009).

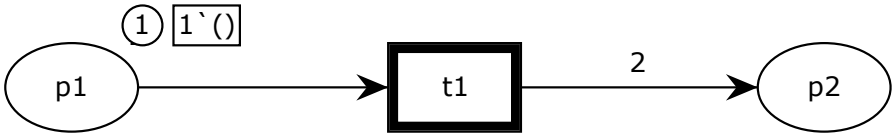
As PNs são uma ferramenta matemática e gráfica, para modelagem, análise, controle, validação e implementação de muitos sistemas, especialmente sistemas que possam ser interpretados como sistemas a eventos discretos (MURATA, 1989; PETERSON, 1981).

Uma RP é um tipo de grafo bipartido e direcionado, em que os arcos nunca ligam dois nós do mesmo tipo. Na Figura 1 são apresentados os elementos gráficos que compõem uma rede de Petri.

Na Figura 1, p_1 e p_2 são os lugares, t_1 é a transição. Neste caso, p_1 é um lugar de entrada de t_1 e p_2 é um lugar de saída de t_1. O arco que liga p_1 a t_1 possui peso 1. O arco que liga t_1 a p_2 possui peso 2. O lugar p1 possui uma ficha, representada na Figura pelo valor "1" próximo ao lugar.

Nas RP, a ocorrência de um evento está associada ao disparo de uma transição e os lugares de entrada e saída da transição representam, respectivamente, as pré-condições e pós-condições associadas à ocorrência do evento. Os arcos de entrada de uma transição t_j têm origem em um ou mais lugares de entrada p_i de t_j e terminam na transição t_j; os arcos de saída têm origem na transição t_j e terminam em um ou mais lugares de saída p_i de t_j. Conforme a

Figura 1 – Ilustração de uma RP



Fonte: Elaborada pelos autores.

Figura 1, o lugar p_1 é entrada de t_1, visto que um arco se origina em p_1 e termina em t_1. O lugar p_2 é saída de t_1, visto que um arco de peso 2 se origina em t_1 e termina em p_2.

As fichas são usadas nas RPs para modelar a dinâmica e as atividades concorrentes do sistema. O estado de uma RP é representado por um número k_i de fichas contidas em cada lugar p_i, chamada marcação, conforme apresentado na Figura 1. O estado do sistema é dado pela distribuição de fichas nos lugares da RP e cada lugar representa um estado parcial do sistema. A mudança de estado é representada pelo movimento de fichas na RP, que acontece quando ocorre o disparo de transições. Cada evento que ocorre no sistema é associado ao disparo de uma transição no modelo RP. O disparo de uma transição significa que o seu evento correspondente ocorreu.

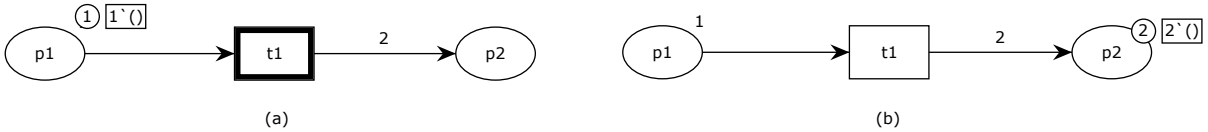
Uma transição é dita habilitada se cada lugar de entrada da transição contém um número de fichas maior ou igual ao peso do arco que o conecta à transição. Uma transição habilitada pode ou não disparar. Quando ocorre o disparo de uma transição, fichas são removidas dos lugares de entrada da transição e fichas são adicionadas aos lugares de saída. A quantidade de fichas removidas e acrescentadas depende do peso do arco. A nova marcação resultante do disparo da transição representa o novo estado do sistema. A marcação inicial M_0 representa o estado inicial da RP.

A definição formal de uma RP é apresentada a seguir (MURATA, 1989).

Uma rede de Petri é uma 5-upla, $PN = (P, T, F, W, M_0)$ em que:

- $P = \{p_1, p_2, .., p_m\}$ é um conjunto finito de lugares;
- $T = \{t_1, t_2, ..., t_n\}$ é um conjunto finito de transições;
- $F \subseteq (P \times T) \cup (T \times P)$ é um conjunto de arcos (fluxo de relações);
- $W : F \rightarrow \{1, 2, 3...\}$ é uma função peso;
  - w(p, t) peso do arco que liga o lugar à transição;
  - w(t, p) peso do arco que liga a transição ao lugar;
- $M_0 : P \rightarrow N^*$ é a marcação inicial, em que N denota os números naturais e M_0 a marcação inicial;
- $P \cap T = \emptyset$ e $P \cup T \neq \emptyset$

Figura 2 – Representação do disparo da transição t_1. (a) estado inicial da RP antes do disparo, (b) estado da RP após o disparo.



Fonte: Elaborada pelos autores.

O comportamento dinâmico das RPs obedece à regra de disparo de transições, a saber:

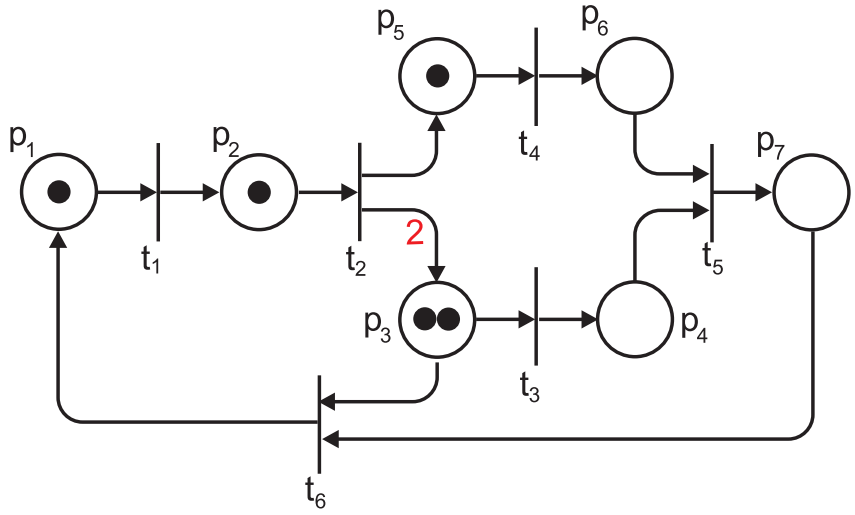
- uma transição é dita habilitada se cada lugar de entrada p de t é marcado com pelo menos $w(p,t)$ fichas, em que $w(p,t)$ é o peso do arco de p para t;
- uma transição habilitada pode ou não disparar;
- o disparo de uma transição t remove $w(p,t)$ fichas de cada lugar de entrada p de t, e adiciona $w(t,p)$ fichas a cada lugar de saída p de t, em que $w(t,p)$ é o peso do arco direcionado de t para p.

A RP apresentada na Figura 2 ilustra a regra de disparo através da modelagem do comportamento dinâmico de um sistema. A Figura 2(a) apresenta o estado inicial do sistema. A mudança de estado, que acontece através do disparo da transição t_1, é apresentada na Figura 2(b). Na Figura 2(a), existe uma ficha no lugar de entrada p_1 e nenhuma ficha no lugar de saída p_2. A marcação da rede é $M_0 = (1,0)$. Nesta marcação a transição t_1 está habilitada e pode disparar. Conforme apresentado na Figura 2(b), no disparo da transição t_1, uma ficha é removida do lugar de entrada p_1 e duas fichas são adicionadas ao lugar de saída p_2, originando uma nova marcação ou estado do sistema $M_1 = (0,2)$. Como pode ser observado neste exemplo, a quantidade de fichas removidas do lugar de entrada e adicionada ao lugar de saída depende do peso dos arcos.

Outro exemplo de uma estrutura PN é dado a seguir

```
PN = (P,T,I,O,W,Mo)
P  = {p1,p2,p3,p4,p5,p6,p7}
T  = {t1,t2,t3,t4,t5,t6}
I(t1) = {p1}          O(t1) = {p2}
I(t2) = {p2}          O(t2) = {p3,p4}
I(t3) = {p3}          O(t3) = {p5}
I(t4) = {p4}          O(t4) = {p6}
I(t5) = {p5,p6}       O(t5) = {p7}
I(t6) = {p3,p7}       O(t6) = {p1}
W ={[(p1,t1),1], [(t1, p2),1], [(p2,t2),1], [(t2,p3)
   ,2],
[(t2,p4),1],  [(t3,p3),1], [(t3,p5),1], [(t4,p4),1],
[(t4,p6),1], [(t5,p5),1], [(t5,p6),1], [(t5,p7),1],
[(t6,p3),1], [(t6,p7),1], [(t6,p1),1]}
M_o = {(p1,1),(p2,1),(p3,2),(p4,0), (p5,1),(p6,0),(p7
   ,0)}
```

Figura 3 – Um grafo PN equivalente à estrutura PN que acabamos de descrever.



Fonte: Elaborada pelos autores.

O modelo gráfico equivalente à estrutura anterior é representado [1] na Figura3.

Observa-se que há sete lugares representados por círculos, seis transições representadas por barras, seis arcos que conectam lugares a transições (arcos de entrada - I) e seis arcos que conectam transições a lugares (arcos de saída - O). O arco de saída que liga a transição 2 ao lugar 3 tem peso igual a 2 e os demais pesos são iguais a 1. Vale ressaltar que os pesos são representados próximos aos arcos. Quando um determinado arco não possui número, o peso é igual a 1. Além disso, a marcação inicial $(1,1,2,0,1,0,0)$ refere-se ao número de fichas em cada lugar da rede.

2.1.1 Propriedades e métodos de análise de rede de Petri

A aplicação de RP na modelagem de sistemas tem a vantagem de permitir verificar as propriedades dos modelos construídos através dos métodos de análise formais, a saber: Árvore (Grafo) de Alcançabilidade ou Cobertura, Matriz de Incidência e Equação de Estado e Técnicas de Redução e Decomposição (MURATA, 1989; PETERSON, 1981). A análise das propriedades das RP com estes métodos pode revelar informações importantes sobre a estrutura e comportamento do sistema modelado, permitindo ao projetista realizar modificações e as correções antes da implementação. Algumas destas propriedades são:

- (Liveness – vivacidade) permite saber se um sistema não possui bloqueio (deadlock) e se todos os elementos do sistema estão ativos;

[1] Todos os modelos mostrados nas figuras foram desenvolvidos usando o simulador CPN Tools. Portanto, todos os códigos usam a sintaxe da linguagem CPN ML para especificar declarações e inscrições de rede. Esta linguagem é uma extensão da linguagem de programação funcional Standard ML.

- (Reversibility – reversibilidade ou reinicialização) permite saber se o sistema é capaz de sempre retornar ao estado inicial após uma tarefa realizada;
- (Boundedness – limitação) permite verificar a consistência de um sistema quanto aos limites de sua capacidade de armazenamento e de realização de tarefas.

Em geral, os sistemas do mundo real são complexos e possuem vários processos com características similares, mas não idênticos. As RP possuem apenas um tipo de ficha, que pode ser inteiro ou booleano. O fato das RP não manipularem tipos de dados diferentes, dificulta a modelagem de sistemas reais e complexos (JENSEN; KRISTENSEN, 2009).

Para modelar sistemas reais, muitas vezes é necessário construir várias subredes independentes com estruturas basicamente idênticas para processos similares. Isto pode tornar o modelo RP extremamente grande, dificultando o desenvolvimento do projeto e a visualização dos modelos na sua totalidade. Além disso, pode ser difícil observar similaridades e diferenças entre as redes individuais que representam as partes similares. Outro fato é que as RP não tratam de restrições de tempo, características inerentes aos sistemas reais.

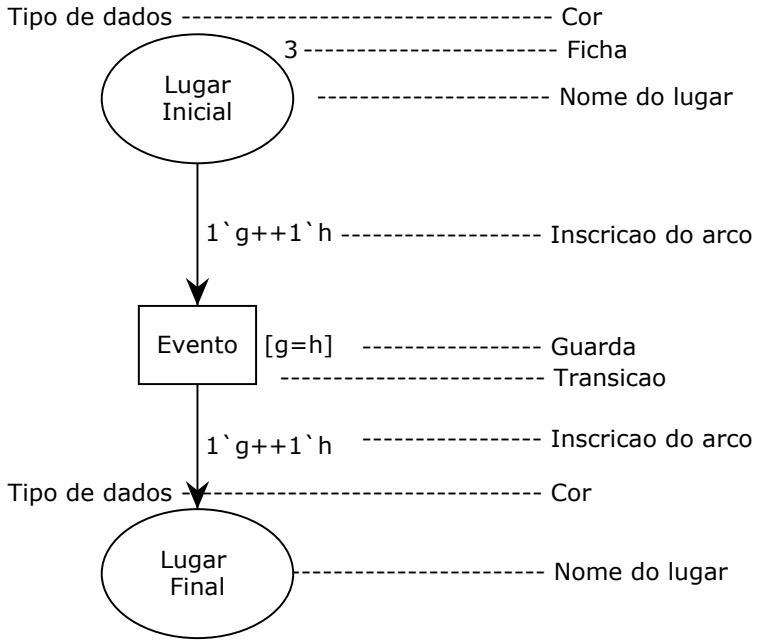
Para contornar estes problemas foram desenvolvidas redes de Petri de alto nível, capazes de descrever sistemas mais complexos de forma mais compacta, elas são: as redes de Petri coloridas, temporizadas e hierarquicas (JENSEN; KRISTENSEN, 2009).

2.1.2 Redes de Petri Coloridas

As *redes de Petri coloridas* (RPC) são uma linguagem gráfica para a construção de modelos de sistemas a eventos discretos e análise de suas propriedades. RPC são uma linguagem de modelagem que combina as capacidades das redes de Petri com os recursos de uma linguagem de programação de alto nível (JENSEN; KRISTENSEN, 2009). A linguagem de programação das RPC é a CPN ML, que se baseia na linguagem de programação funcional Standard ML (ULLMAN, 1998). RPC são destinadas ao uso prático, em especial, porque elas permitem a construção de modelos compactos e paramétricos. A vantagem das RPC sobre as outras redes de Petri é a capacidade de modelar sistemas complexos e fornecer modelos com um alto nível de abstração (JENSEN; KRISTENSEN, 2009).

O tempo desempenha um papel significativo numa vasta gama de sistemas concorrentes. O funcionamento correto de alguns sistemas depende crucialmente do tempo tomado por certas atividades, e diferentes decisões de projeto podem ter um impacto significativo sobre o desempenho de um sistema. RPC incluem um conceito de tempo que faz com que seja possível

Figura 4 – Representação de uma Rede de Petri Colorida



Fonte: Elaborada pelos autores.

capturar o tempo tomado por eventos no sistema. Isto significa que as RPC podem ser aplicadas para a análise de desempenho baseada em simulação para investigar métricas, tais como atrasos, taxa de transferência e tamanho de fila, e para a modelagem e validação de sistemas de tempo real.

Modelos RPC podem conter diferentes níveis de abstração. Podem ser construídos em um conjunto de módulos, que são importantes quando lidamos com modelos de sistemas de grande porte. Os módulos interagem uns com os outros através de um conjunto de interfaces bem definidas, de uma forma semelhante às linguagens de programação. O conceito de módulos em RPC baseia-se num mecanismo de estruturação hierárquica, o que permite que um módulo tenha submódulos que podem ser reutilizados em diferentes partes do modelo.

O CPN Tools é uma ferramenta para edição, simulação, análise de espaço de estado, e análise de desempenho de modelos RPC. O usuário do CPN Tools trabalha diretamente com a representação gráfica do modelo RPC CPNTOOLS (2021).

Na Figura 4 é mostrada a estrutura principal de um modelo RPC, constituído por lugares, transições e arcos. Com RPC, é possível usar os tipos de dados e manipulação de dados complexos.

Os lugares representam o estado do sistema modelado. Os nomes dos lugares são escritos dentro das elipses, e não têm nenhum significado formal - mas eles têm enorme importância prática para a legibilidade de um modelo RPC. Cada lugar pode ser marcado com

uma ou mais fichas, e cada ficha tem um valor de dados ligado a ela. Este valor de dados é chamado cor da ficha. É o número de fichas e as cores das fichas nos lugares individuais que, em conjunto, representam o estado do sistema. Isto é chamado uma marcação do modelo RPC. As fichas em um lugar específico constituem a marcação daquele lugar.

As transições representam os eventos que podem ocorrer no sistema. Tal como acontece com os lugares, os nomes das transições são escritos dentro dos retângulos. Quando a transição ocorre, ela remove fichas de seus lugares de entrada (aqueles lugares que têm um arco que conduz à transição) e adiciona fichas aos seus lugares de saída (os lugares que têm um arco proveniente da transição). As fichas que são removidas dos lugares de entrada e adicionadas aos lugares de saída, quando ocorre uma transição, são determinadas por meio das expressões de arco, que são as inscrições textuais posicionadas ao lado dos arcos individuais. As expressões de arco são escritas na linguagem de programação CPN ML e são construídas a partir de variáveis, constantes, operadores e funções. Quando todas as variáveis em uma expressão são ligadas a valores do tipo correcto, a expressão pode ser avaliada. As transições possuem uma guarda, que é uma expressão booleana. Quando uma guarda está presente, ela deve ser avaliada como verdadeira para que a transição possa ser habilitada, caso contrário, a transição estará desabilitada e não pode ocorrer. Assim, uma guarda coloca uma restrição adicional sobre a habilitação da transição.

2.1.3 Rede de Petri Temporizada

Quando modelamos sistemas que possuem restrições de tempo, podemos atribuir um valor de tempo às transições. Esse modelo usa o conceito de tempo para representar a duração de ações, tarefas e eventos em um sistema real.

Em um modelo RPC temporizado, medidas de desempenho, tais como comprimento máximo da fila e média de tempo de espera, podem ser calculadas. Podemos verificar, também, se a operação de um sistema em tempo real obedece aos prazos exigidos. As fichas podem transportar um selo de tempo, além de sua cor. Isto significa que a marcação de um lugar em que as fichas possuem selo de tempo é agora um multiconjunto temporizado. Além disso, o modelo RPC possui um relógio global que representa o tempo de modelo. A distribuição das fichas nos lugares, em conjunto com as suas temporizações e o valor do relógio global, é chamada de marcação temporizada. Mesmo num modelo RPC hierárquico temporizado há um único relógio global. O selo de tempo especifica o tempo em que a ficha está pronta para ser utilizada, ou

removida por uma transição que ocorre.

Retardos de tempo podem ser implementados nos arcos e nas transições. Se houver um tempo de retardo em uma transição, a sua ocorrência adiciona o valor do retardo a todas as fichas de saída que carregam um selo de tempo. Se há um tempo de retardo associado a um arco de saída de uma transição, o retardo é adicionado apenas para as fichas temporizadas que são colocadas no lugar de saída associado a esse arco.

No modelo proposto por esta tese, usamos a RPC temporizada para que o conjunto de dados seja apresentado a rede através de épocas, uma época é a apresentação de todas as amostras do conjunto de dados uma vez.

2.1.4 Rede de Petri Hierárquica

O modelo hierárquico permite a construção de modelos mais complexos, temos o modelo principal e nele, transições de substituição associadas a sub-redes que são executadas na ocorrência dessas transições. As redes de Petri hierárquicas coloridas temporizadas permitem a descrição de sistemas múltiplos que têm interações complexas entre componentes, preservando o paralelismo natural do comportamento do sistema. Isso é altamente conveniente para a implementação de processos paralelos, ou arquiteturas de fluxo de dados de computadores.

2.1.5 Linguagens Funcionais

A linguagem utilizada é a linguagem CPN ML, esta linguagem é uma extensão da linguagem de programação funcional Standard ML. As linguagens funcionais são uma classe de linguagens projetadas para refletir a maneira como as pessoas pensam matematicamente, tratando a computação como uma avaliação de funções matemáticas e que evita estados ou dados mutáveis. Em contraste com as linguagens imperativas usuais, nas quais as variáveis representam células na memória que podem ser modificadas usando o operador de assinatura $=$ (ou $:=$), as linguagens funcionais veem o uso do operador como uma expressão de uma equação. Por exemplo, se um programa funcional contiver a declaração

$$let(x) = f(y) \tag{2.1}$$

então isso introduz o nome x e afirmaria que a equação $x = f(y)$ é verdadeira. Não há uma célula de memória e certamente x não pode mudar (de modo que a equação se tornasse falsa). As funções são tratadas como valores de primeira ordem, o que é o mesmo que dizer que funções

podem ser parâmetros ou valores de entrada para outras funções e podem ser os valores de retorno ou saída de uma função. Então podemos entender o paradigma funcional como um mapeamento dos valores de entrada nos valores de retorno, através de funções.

Como as variáveis em um programa funcional não podem ser modificadas, a repetição deve ser expressa em um programa funcional por meio de recursão, em vez do uso de loops. Funções recursivas invocam a si mesmas, permitindo que uma operação seja realizada várias vezes. Recursividade em programação funcional pode assumir várias formas e é em geral uma técnica mais poderosa que o uso de laços. A grande vantagem da recursão está na possibilidade de usar um programa de computador finito para definir, analisar ou produzir um estoque potencialmente infinito de sentenças, designs ou outros dados.

2.2 Redes Neurais Artificiais

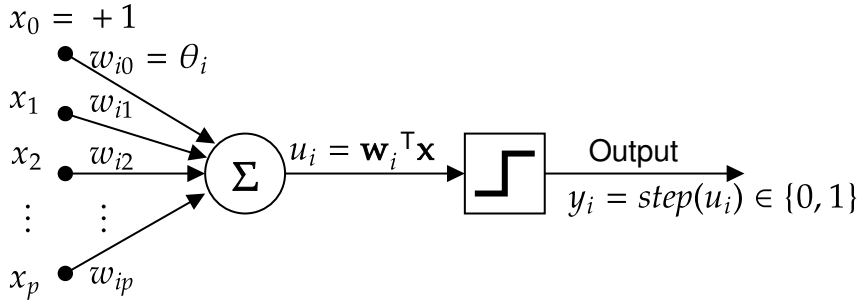
As RNAs tem obtido resultados impressionantes em diversas áreas do conhecimento, elas imitam a ação de uma rede biológica de neurônios. Nesse cenário, iniciaremos nossa abordagem com o neurônio biológico e seu modelo matemático, chamado de modelo de McCulloch-Pitts. Em sequência, descreveremos a rede perceptron simples, que é a combinação do neurônio de McCulloch-Pitts e uma regra de aprendizado e por último, o perceptron multicamadas juntamente com o algoritmo de treinamento backpropagation.

2.2.1 O Neurônio Artificial de McCulloch-Pitts

O neurônio é responsável pelo processamento e fluxo de todas as informações no cérebro humano. Ele está dividido estruturalmente em quatro partes: dendritos, sinapses, corpo celular e axônio. Os dendritos são ramificações correspondentes aos canais de entrada de informação (sinais elétricos); as sinapses são pontos de contato entre neurônios onde há passagem de neurotransmissores do axônio de um neurônio para os dendritos de outro neurônio; o corpo celular é o local onde é feito o balanço energético da célula nervosa (soma das contribuições de energia); o axônio é o canal de saída do neurônio, ou seja, caminho de propagação dos impulsos nervosos em direção a outros neurônios ou músculos. Quando o balanço energético no corpo celular for maior que um certo limiar, o axônio emite um impulso elétrico ou potencial de ação, a emição deste impulso é o disparo do neurônio. As sinapses podem ser excitatórias (facilitam a passagem do potencial de ação) ou inibitórias (inibem a passagem do potencial de ação).

Um modelo matemático do neurônio foi proposto por McCulloch e Pitts (1943) serve

Figura 5 – Esboço do modelo de neurônio de McCulloch-Pitts usado nesta tese e adaptado de (MCCULLOCH; PITTS, 1943)



Fonte: Elaborada pelos autores.

até hoje como bloco construtivo básico de algoritmos de redes neurais. A seguir, apresentamos a modelagem do processamento de informações de um neurônio. As entradas $\{x_1, x_2, \ldots, x_p\}$, $x_j \in \mathbb{R}$. são ponderadas por pesos sinápticos que podem ser excitatórios (positivos) ou inibitórios (negativos) $\{w_{i1}, w_{i2}, \ldots, w_{ip}\}$, $w_{ij} \in \mathbb{R}$ e adicionada ao limiar $\theta_i \in \mathbb{R}$. Formalmente, essa operação definida como ativação do i-th neurônio como

$$u_i = \sum_{j=1}^{p} w_{ij} x_j + \theta_i = \sum_{j=0}^{p} w_{ij} x_j, \tag{2.2}$$

em que inserimos o limiar θ_i no segundo somatório definido $w_{i0} = \theta_i$ e $x_0 = +1$. Esta forma alternativa de escrever a ativação u_i nos permite entendê-lo como o produto escalar $u_i = \mathbf{w}_i^T \mathbf{x}$ entre o vetor de pesos $\mathbf{w}_i \in \mathbb{R}^{p+1}$ e o vetor de entrada $\mathbf{x} \in \mathbb{R}^{p+1}$. Para esta proposta, os vetores $\mathbf{x}$ and $\mathbf{w}_i$ devem ser definidos respectivamente como $\mathbf{x} = [x_0\ x_1\ x_2\ \ldots\ x_p]^T$ and $\mathbf{w}_i = [w_{i0}\ w_{i1}\ w_{i2}\ \ldots\ w_{ip}]^T$.

Finalmente, a função degrau $g(\cdot)$ é aplicado a u_i para produzir a saída do i-ésimo neurônio

$$y_i = g(u_i) = g(\mathbf{w}_i^T \mathbf{x}) = g\left(\sum_{j=1}^{p} x_j w_j + \theta_i\right), \tag{2.3}$$

onde $g(u_i) = 1$, se $u_i > 0$, e $g(u_i) = 0$, caso contrário.

2.2.2 A Rede Perceptron Simples

O neurônio possui modelagem matemática simples, mas quando reunimos mais de um neurônio em camadas, esses neurônios têm poderosas propriedades computacionais e isso levou à rápida implantação de várias arquiteturas de RNA. A primeira foi o perceptron simples, que une o modelo McCulloch-Pitts com uma regra de aprendizado, tornando a rede inteligente.

O processo de aprendizado consiste principalmente em calcular o erro, que é a comparação entre a saída desejada e a saída gerada pela rede. De acordo com esse erro, os pesos e limiares são modificados até que o problema de interesse seja resolvido ou que o período de aprendizagem tenha finalizado.

Definindo algumas variáveis a serem utilizadas, o vetor de entrada no instante t é

$$\mathbf{x}(t) = [x_0(t)\ x_1(t)\ x_2(t)\ \ldots\ x_p(t)]^T, \mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \eta e_i(t)\mathbf{x}(t) \tag{2.4}$$

A Equação que ajusta pesos e limiares, dado o vetor de entrada da Equação (2.4)

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \eta e_i(t)\mathbf{x}(t) \tag{2.5}$$

Em que

- $\mathbf{w}_i(t) \in \mathbb{R}^{p+1}$ é o vetor peso do i-ésimo neurônio;
- $\eta \in (0,1)$ é a taxa de aprendizado;
- $e_i(t) = d_i(t) - y_i(t)$ é o erro do i-ésimo neurônio no tempo t, $d_i(t)$ é a saída desejada para o i-ésimo neurônio e $y_i(t)$ é a saída gerada por esse neurônio.
- $\mathbf{x}(t) \in \mathbb{R}^{p+1}$ é o vetor de entrada.

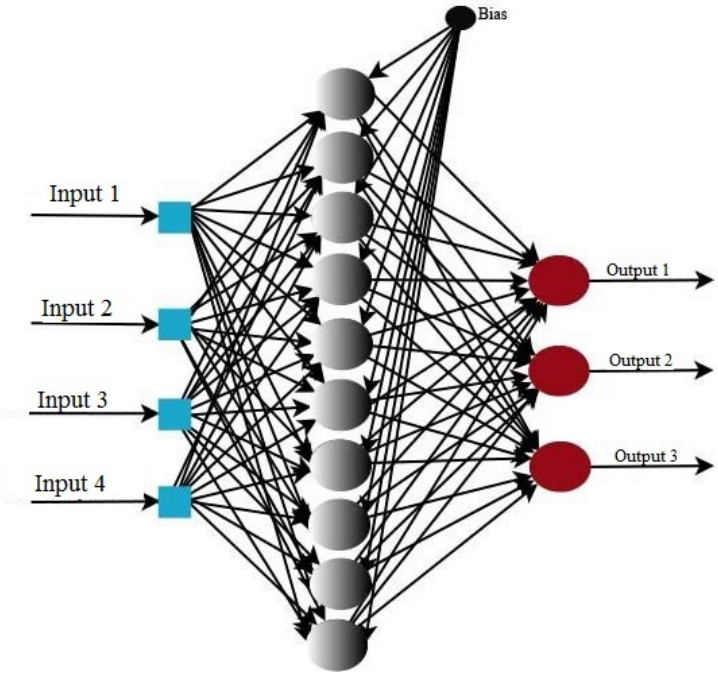
2.2.3 A Rede MLP e o Algoritmo de Retropropagação do Erro

Como descrito na seção anterior, o perceptron resolve com eficiência problemas linearmente separáveis, mas em muitos problemas práticos as classes não são linearmente separáveis. Para resolver esses problemas, foram incluídas mais camadas no perceptron, porque cada camada processa os dados de forma não-linear com o intuito de obter uma representação (projeção) que facilite a separação das classes para os neurônios da camada de saída. Assim, obtém-se a MLP, com uma camada de entrada, uma ou mais camadas ocultas (o número de camadas ocultas depende da complexidade do problema) e uma camada de saída, conforme mostrado na Figura 6.

Uma outra mudança é na função de ativação, enquanto que o os neurônios da rede PS usam a função step, os neurônios da rede MLP devem usar uma função de ativação sigmoidal, tal como a função tangente hiperbólica (tanh), cuja cuja a expressão é mostrada a seguir:

$$y_i(t) = \frac{1 - \exp(-u_i(t))}{1 + \exp(-u_i(t))} \in (-1, +1). \tag{2.6}$$

Figura 6 – Exemplo de uma arquitetura da rede MLP com 4 entradas, 3 saídas e 1 camada oculta com 10 neurônios.



Fonte: Elaborada pelos autores.

Por sua vez, a derivada da função tanh é dada por

$$y_i'(t) = \frac{dy_i(t)}{du_i(t)} = 0,5[1 - y_i^2(t)] \tag{2.7}$$

O método dominante para o treinamento destas redes é o algoritmo de retropropagação do erro (WERBOS; JOHN, 1974; RUMELHART *et al.*, 1986; CUN, 1986; PARKER, 1985), um algoritmo eficiente para calcular o gradiente da função de perda, que quando combinado com o gradiente descendente estocástico pode modificar cada peso da rede neural a fim de reduzir o erro médio quadrático da tarefa de aproximação de função.

2.3 Revisão Bibliográfica

Estes trabalhos serão discutidos a seguir, principalmente em relação à capacidade de aprendizado existente nos modelos propostos. As redes de Petri foram utilizadas para modelar o neurônio biológico em (AHSON, 1995), (HABIB; NEWCOMB, 1990b) , (CHOW; LI, 1997) e (KORIEM, 2001b).

O modelo *Neuron Type Processor Modeling Using a Time Petri Net* (NTP), propost por Habib e Newcomb (1990b) (HABIB; NEWCOMB, 1990b) utiliza uma rede temporizada para modelar o neurônio. No modelo, a soma do peso pela única entrada é indicado por uma

sequência de disparo de transições, e a comparação dessa soma com o limiar é modelado com o disparo de uma transição, atribuindo assim a saída da rede, no entanto, foi apenas indicado, essa operação não é realizada no modelo e nem implementada (ou simulada).

O modelo *Higher-Order Petri Net* (HOPN), propost por Chow e Jin-Yan Li (CHOW; LI, 1997) modela o neurônio biológico de uma maneira mais elegante, utilizando arcos de ordem superior (*higher-order*). Entretanto, como no modelo NTP as operações foram apenas indicadas, não havendo implementação computacional do modelo HOPN.

O modelo de Ahson (AHSON, 1995), utiliza o modelo NTP para o neurônio biológico (HABIB; NEWCOMB, 1990b) e modela o perceptron com dois neurônios e duas entradas, chamando esse modelo de *Adaptive Neural Processor* (ANP). Como nos demais modelos supracitados as operações não são modeladas, somente indicadas no modelo.

O modelo CN-net, proposto por KORIEM (2001b),(KORIEM, 2001b) elevou o nível das redes de Petri usadas para emular arquiteturas de redes neurais, pois utiliza redes de Petri coloridas. Inicialmente, modelou o neurônio artificial e depois os autores estendem este modelo para uma rede feedforward para o problema do XOR. É uma rede mais elegante, que consegue receber entradas de diferentes tipos, porém como as demais citadas, as funções são apenas indicadas, elas não são realizadas e implementadas no modelo. A CN-net consegue fazer uma melhor análise das propriedades de uma rede neural, porém não foi feito um quadro completo de capacidade de aprendizado das arquiteturas resultantes, tais como uma regra para atualização dos pesos e limiares, bem como uma estrutura geral e reutilizável, pois para cada problema (banco de dados), precisa ser construída uma nova rede de Petri. Além disso, uma mudança na arquitetura da rede neural (acréscimo ou decréscimo de neurônios) acarreta uma mudança na estrutura gráfica da rede de Petri.

Com base na limitada capacidade de representação do conhecimento e na ausência de aprendizado dos modelos previamente descritos, esta tese desenvolverá um modelo capaz de representar em sua totalidade uma rede neural MLP, capaz de aprender tarefas complexas via algoritmo de retropropgação do erro e também ser reutilizada em diferentes problemas de classificação e regressão.

Este capítulo, mostrou uma revisão bibliográfica sobre redes de Petri e redes neurais artificiais. No próximo capítulo, será apresentado os modelos propostos por esta tese de rede de Petri coloridas isomórficas às redes neurais.

3 REDES DE PETRI COLORIDAS ISOMÓRFICAS ÀS REDES NEURAIS

Neste capítulo, mostra-se a implementação completa das redes de Petri coloridas desenvolvidas nesta tese que são isomórficas ao neurônio artificial de McCulloch-Pitts, à rede Perceptron Simples e à MLP, como também a análise das mesmas com o banco de dados Iris para fins didáticos.

3.1 Modelo McCulloch-Pitts-CPN

Após uma breve revisão das redes de Petri coloridas realizada na Seção 2.1 e do modelo McCulloch-Pitts na Seção 2.2, temos o embasamento teórico suficiente para descrevermos o modelo MP-CPN desenvolvido nesta tese. Na Figura 7 a RPC isomórfica (MP-CPN). As fichas no lugar *input* da MP-CPN contém a marcação que representa as entradas $x_1, x_2, ..., x_n$, (no exemplo da Figura 7, $n = 7$). As fichas no lugar *weights* da RPC, as quais possuem três campos ($m = 3$), contêm a marcação que representa os pesos $w_1, w_2, ..., w_n$. A função u_i da MP-CPN modela o somatório. Esta função é associada ao arco que liga a transição *sum* ao lugar *Ui*. A implementação desta função está descrita a seguir.

```
fun u_i(u1,u2,u3,bias) =
let
       val(x0,w0) = bias
       val(x1,w1) = u1
       val(x2,w2) = u2
       val(x3,w3) = u3
in
       ((x0*w0)+(x1*w1)+(x2*w2)+(x3*w3))
end
```

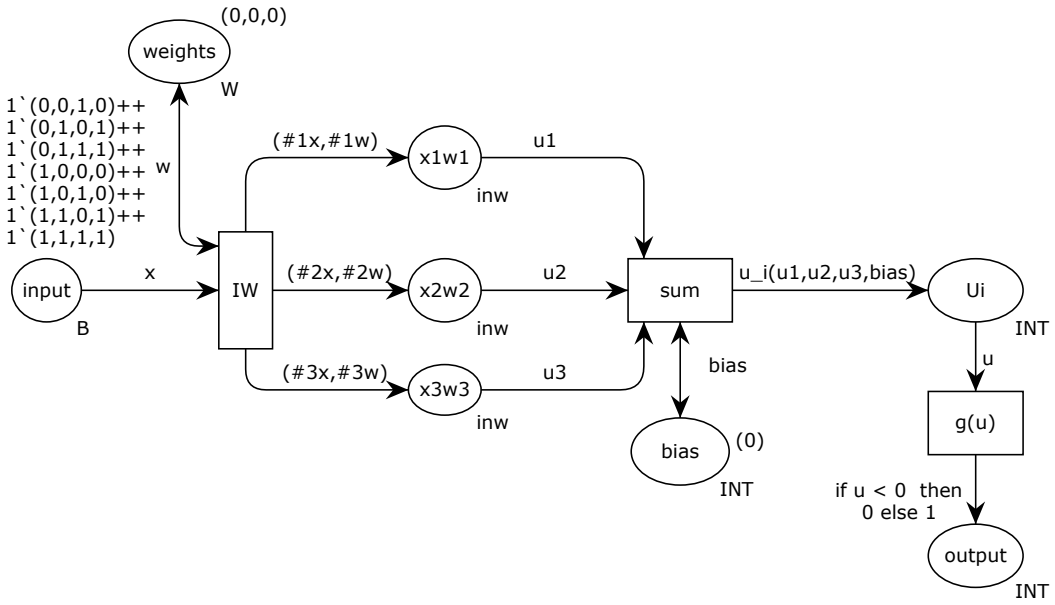
A função step da Figura 5 é modelada pela função [if $u < 0$ then 0 else 1], associada ao arco que liga a transição $g(u)$ ao lugar *output* na RPC.

3.2 Modelo Perceptron-CPN

Com a proposição do neurônio artificial de McCulloch-Pitts em 1943, Rosenblatt (1958) propõe a rede Perceptron simples (PS) (ROSENBLATT, 1958), que é a união do neurônio MP com uma regra de aprendizado. É esta regra que torna a rede PS um dispositivo inteligente, por isso a rede PS é considerada o primeiro algoritmo de redes neurais artificiais.

O modelo Perceptron-CPN, mostrado na Figura 8 foi desenvolvido com base no MP-CPN, adicionando a regra de aprendizagem. Utiliza-se do modelo MP-CPN o somatório

Figura 7 – Modelo MP-CPN



Fonte: Elaborada pelos autores.

do produto de pesos e entradas, função *summation* e saída *y* com a função step. Depois disso, calculamos o erro

$$e = S_{Desired} - S_{Generated} \tag{3.1}$$

em que $S_{Generated}$ é a saída gerada pela rede e $S_{Desired}$ é a saída desejada.

O cálculo do erro é implementado no quinto campo na função do arco $(\#1b, \#2b, \#3b, \#4b, \#4b - pred)$, que liga a transição *Error* ao lugar *error*, em que #4b é a variável que representa a saída gerada pela rede e *pred* é a saída desejada.

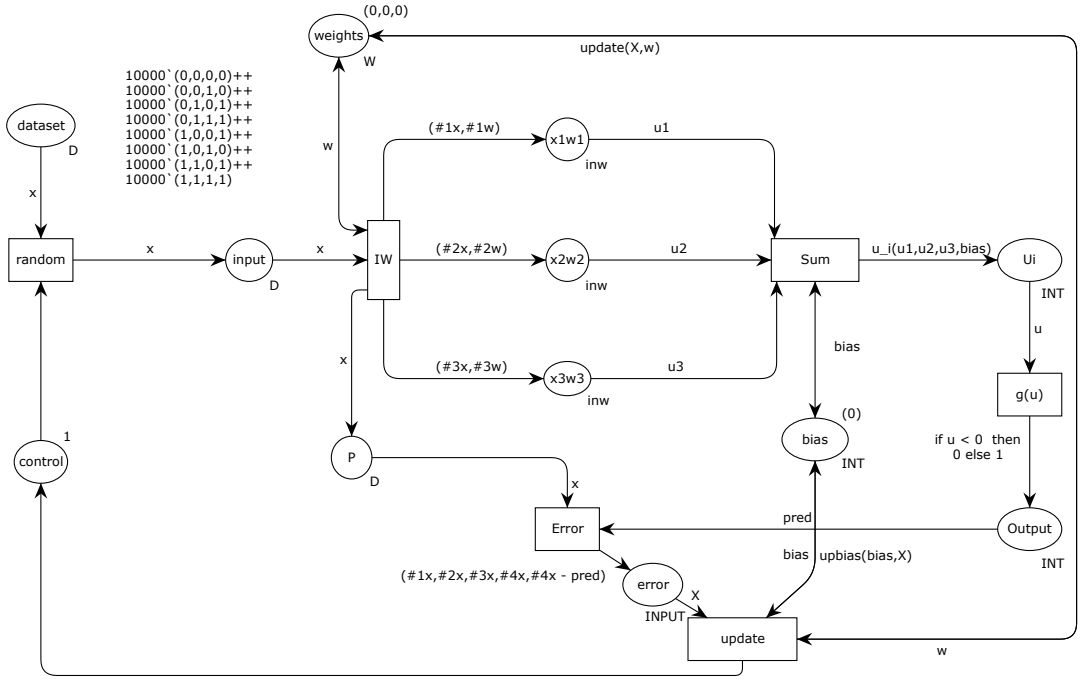
Após o cálculo do erro, atualizamos os pesos com a função *update*, a implementação desta função no Perceptron-CPN é mostrada a seguir.

```
fun  update (x,w) =
     let
         val(x1,x2,x3,x4,e) = x
         val(w1,w2,w3) = w
     in
        (w1+(1*e*x1),w2+(1*e*x2), w3+(1*e*x3))
     end
```

De maneira similar, atualizamos o limiar com a função *upbias*, a implementação desta função no Perceptron-CPN é mostrada a seguir.

```
fun  upbias (bias,X) =
     let
```

Figura 8 – Perceptron-CPN



Fonte: Elaborada pelos autores.

```
      val(x1,x2,x3,x4,e) = X
in
      (bias + 1*e*1)
end
```

3.2.1 Prova de conceito 1: Perceptron-CPN

Para testar o modelo perceptron-CPN, utiliza-se um problema linearmente separável, para classificar alunos em graduandos e mestrandos, o conjunto de dados é mostrado na Tabela 1, sendo composto por três atributos e duas classes que assume o valor 0 ou 1. A classe possui valor 0 para alunos de graduação e valor 1 para alunos de mestrado. Em 4 épocas, o perceptron-CPN foi capaz de aprender a classificar corretamente as duas classes, chegando a *erro quadrático médio* (MSE) igual a zero, a curva de aprendizado é mostrada no gráfico da Figura 10. Esse conjunto de dados é apresentado na Figura 9. Nela, as classes formam cubos bem separados e o perceptron-CPN pode traçar infinitos hiperplanos separadores sem dificuldade.

3.2.2 Prova de Conceito 2: Perceptron-CPN

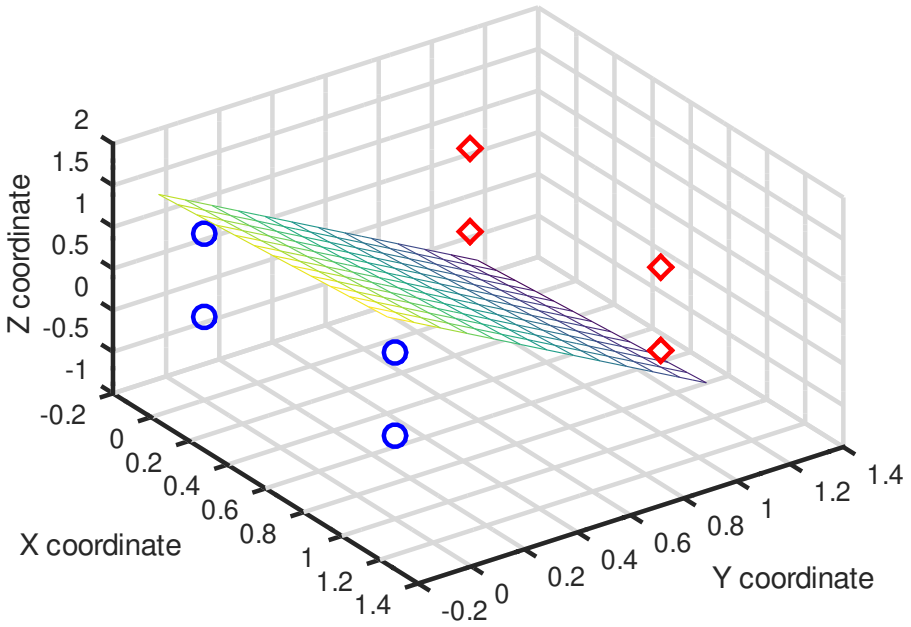
Outro conjunto de dados utilizado para testar o modelo perceptron-CPN é o conjunto IRIS. Ele contém 3 classes de 50 instâncias cada, em que cada classe se refere a um tipo de flor:

Tabela 1 – Conjunto de dados de um problema de classificação binária usado para avaliar o modelo de neurônio Perceptron-CPN proposto. Rótulos: (0) Aluno de Graduação; (1) Aluno de mestrado. Os nomes são fictícios.

Entradas	Nome	Rótulo
000	Isabella	0
001	Ethan	0
010	Emma	1
011	Michael	1
100	Sophia	0
101	Olivia	0
110	Ava	1
111	William	1

Fonte: o autor.

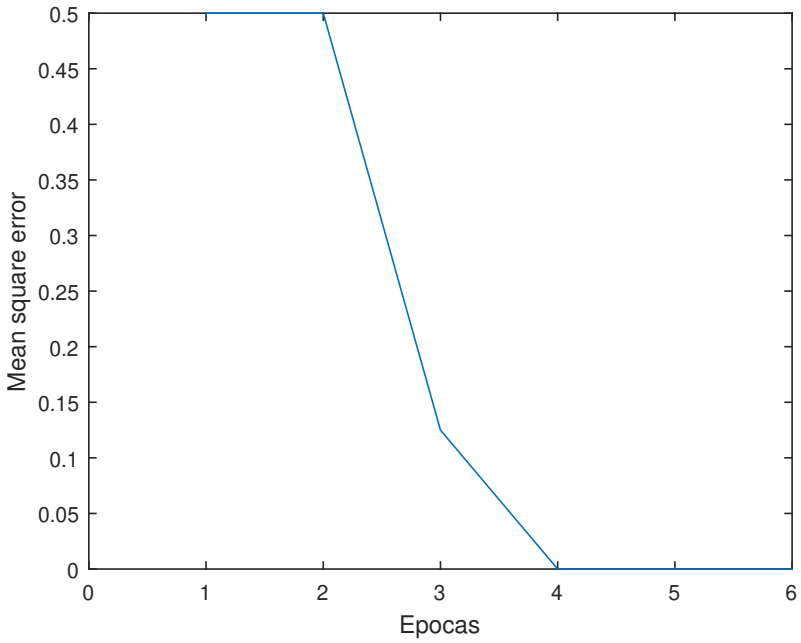
Figura 9 – Hiperplano separador ($w_1x_1 + w_2x_2 + w_3x_3 + w_0 = 0$) associado com os pesos ($w_1 = 0,34, w_2 = 0,22, w_3 = 0,87$) bias ($w_0 = -0,82$) do modelo neural proposto perceptron-CPN depois de um treinamento típico do problema de classificação da tabela 1.



Fonte: Elaborada pelos autores.

iris setosa, iris versicolor e iris virginica. A classe setosa é linearmente separável das outras duas. As classes versicolor e virginica não são linearmente separáveis entre si. Para usar esse conjunto de dados, rotula-se a primeira classe como 0 e as outras duas não linearmente separáveis como 1. Assim, transforma-se esse conjunto de dados em linearmente separável. Em poucas épocas, o modelo de perceptron-CPN aprendeu a separar eficientemente as duas classes. Na Figura 11, mostra-se o modelo de perceptron-CPN aplicado ao conjunto de dados iris. Neste modelo, adiciona-se um lugar, uma transição e uma função para carregar o conjunto de dados de um

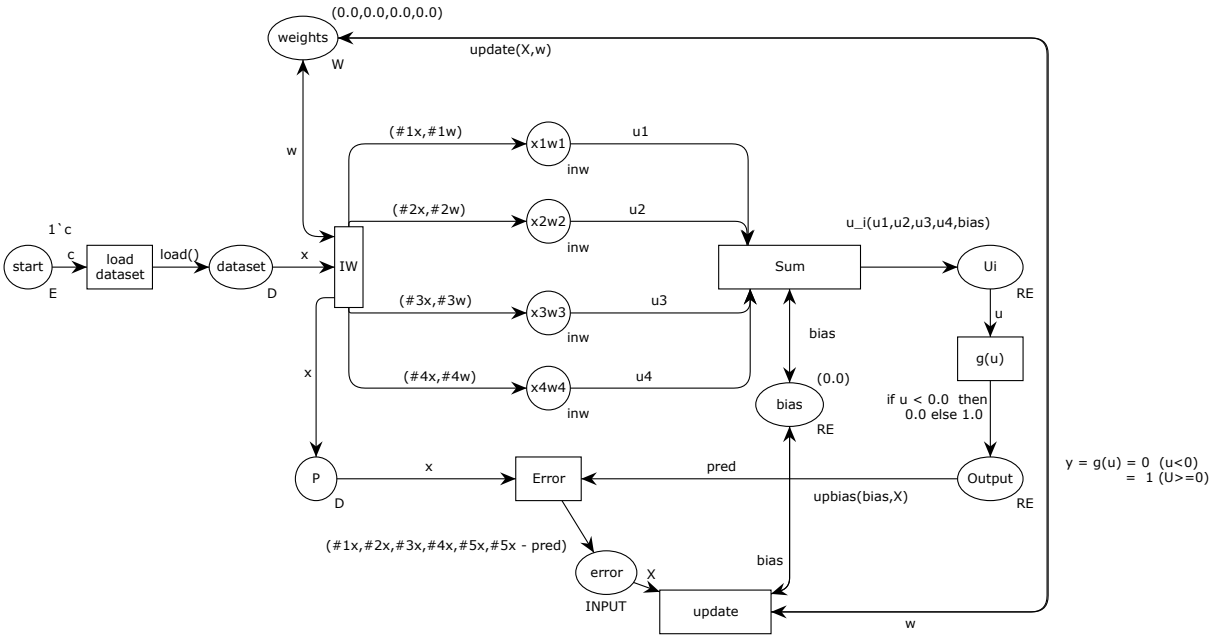
Figura 10 – Curvas de aprendizado do perceptron-CPN para o conjunto estudantes.



Fonte: Elaborada pelos autores.

arquivo com a extensão *txt*, chamada de *load*() (os lugares e a transição estão na cor vermelha). Essa função é geral e carrega qualquer conjunto de dados nesse formato.

Figura 11 – Modelo proposto de rede de Petri para a rede Perceptron (conjunto de dados Iris duas classes)



Fonte: Elaborada pelos autores.

Tabela 2 – Lugares e Transições

Lugares e transições	significado
start $(L)^*$	é o lugar inicial, contém a ficha 1´C que habilita a transição load data
load dataset (T)	o disparo dessa transição carrega o banco de dados e transformando-o em fichas
dataset (L)	lugar que recebe o banco de dados no formato de fichas
random (T)	transição que retira uma tupla (uma linha da tabela ou uma ficha) do banco de dados aleatoriamente
C (L)	lugar que controla a apresentação de uma tupla por vez à rede
input (L)	lugar que recebe a tupla retirada aleatoriamente
W (L)	lugar que contém o conjunto de pesos sinápticos
I (L)	lugar que recebe a saída da ficha
IW (T)	transição que separa os atributos com seus respectivos pesos
x1w1 (L)	lugar que recebe o atributo 1 e o peso 1
x2w2 (L)	lugar que recebe o atributo 1 e o peso 1
x3w3 (L)	lugar que recebe o atributo 1 e o peso 1
sum (T)	esta transição tem a função summation em seu arco, ela recebe os atributos com seus respectivos pesos e bias e realiza o somatório do produto dos mesmos
bias (T)	lugar onde contém uma ficha com o limiar.
Ui (L)	lugar que recebe o somatório do produto dos atributos e pesos
g(u) (T)	esta transição tem a função $if\ u < 0.0\ then\ 0.0\ else\ 1.0$ em seu arco, ela é a função degrau
Yi (L)	lugar que recebe a saída da rede
Error (T)	transição que calcula o erro e o insere no último campo da ficha
error (L)	lugar que recebe a ficha com último campo com erro
update (T)	transição que atualiza os pesos sinápticos

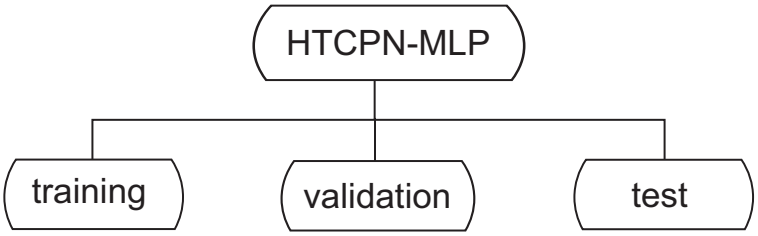
Fonte: o autor.
L : lugar e T: transição

3.3 Rede de Petri Colorida Hierárquica Temporizada Perceptron Multicamadas (HTCPN-MLP)

A rede peceptron-CPN obteve excelentes resultados em problemas linearmente separáveis, porém não consegue resolver problemas não linearmente separáveis como o problema do OU-exclusivo (XOR, na sigla em inglês). Houve um tempo em que a comunidade científica achou que a inteligência artificial não conseguiria resolver esses problemas e somente em 1986 esse problema foi resolvido, quando foi proposto a criação de uma rede com neurônios colocados em camadas, chamada de Perceptron multicamadas (ver Figura 6) e um novo algoritmo de aprendizagem, o algoritmo de retropropagação do erro (RUMELHART *et al.*, 1986). Enquanto o algoritmo de aprendizagem do modelo perceptron-CPN foi obtido através de uma análise geométrica do problema, o o algoritmo de retropropagação do erro foi obtido a partir da minimização de uma função custo baseada no erro médio quadrático. O algoritmo de retropropagação projeta os gradientes locais dos erros dos neurônios de saída para os neurônios da camada oculta.

Na Figura 12 apresentamos um modelo abstrato da HTCPN-MLP contendo a página principal *(HTCPN-MLP)* e três sub-redes chamadas *training*, *validation* e *test*, enquanto que a HTCPN-MLP é mostrada na Figura 13.

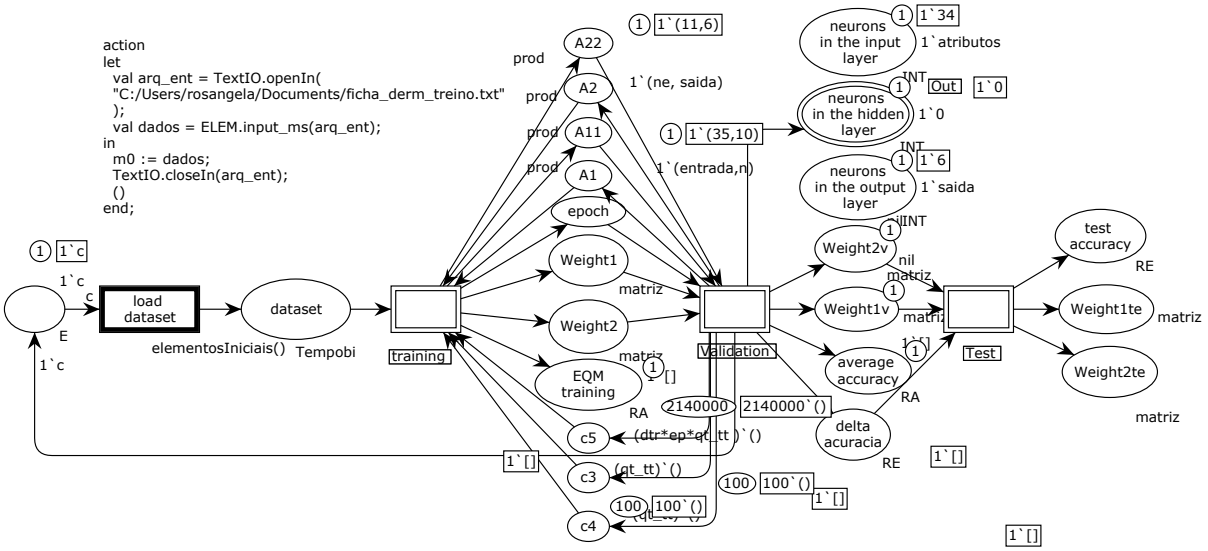
Figura 12 – Modelo Abstrato do HTCPN-MLP



Fonte: Elaborada pelos autores.

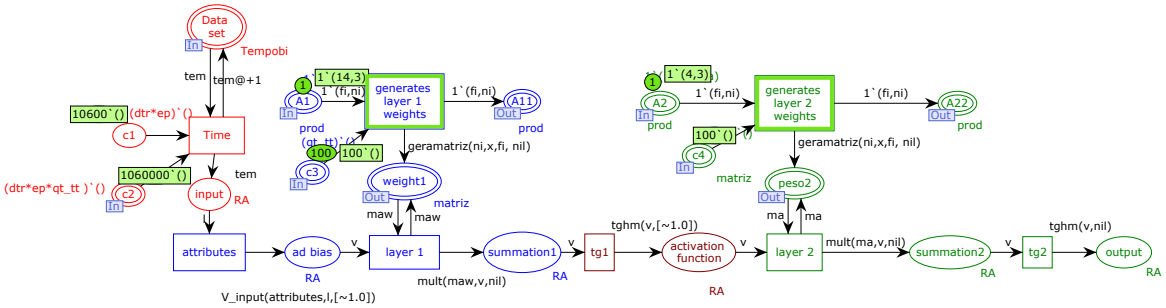
O modelo HTCPN-MLP, implementado no CPN tools e mostrado na Figura 13 tem quatro transições: a primeira transição chamada de *load dataset*, carrega o banco de dados e o envia como fichas para o lugar *dataset*, a segunda transição é de hierarquia chamada de *training*, ela inicia o treinamento da rede, onde as fichas são apresentadas ao HTCPN-MLP com os atributos e saídas desejadas. Com a rede treinada, segue-se para a terceira transição, em que pesos e limiares são enviados para a sub-rede associada à transição de substituição *validation* que modela a validação da rede, a validação consiste em ajustar o processo de treinamento, observando se a rede generaliza bem e tem a quantidade de neurônios adequada para o problema e, finalmente, a quarta transição *test* que é o teste da rede, onde esses pesos e limiares são testados com um conjunto de dados que não está no treinamento para verificar se o HTCPN-MLP é capaz de classificar novos dados.

Figura 13 – Modelo proposto HTCPN-MLP.



Fonte: Elaborada pelos autores.

Figura 14 – Etapa de propagação direta durante o treinamento do modelo proposto HTCPN-MLP.



Fonte: Elaborada pelos autores.

3.3.1 Treinamento do HTCPN-MLP

Na sub-rede *training*, foi implementado o algoritmo retropropagação, que consiste em duas fases: fase de propagação e fase de retropropagação.

1. Fase de propagação (*forward pass*): nesta fase, as entradas são propagadas e as saídas geradas pela rede são obtidas. Esta fase é mostrada na Figura 14 e inicia recebendo o banco de dados no lugar *Data set*, habilitando a transição *Time* que envia as fichas temporizadas para o lugar *input*. A transição *attributes* retira do lugar *input* uma entrada aleatoriamente e a envia para o lugar *ad bias* adicionando o limiar. A transição *layer 1* tem a função de arco *mult* que realiza o somatório do produto das entradas pelos seus respectivos pesos e envia-o para o lugar *summation*. A transição *tg1* envia para o lugar *activation function* o resultado da função de ativação tangente hiperbólica. Repete-se o mesmo procedimento da *layer 1* na *layer 2* e a saída é enviada para o lugar *output*. Foram criados lugares de controle para controlar o número de épocas (C_1, C_2, C_3 e C_4), número de neurônios nas camadas (A_1,A_{11}, A_2,A_{22}) e as transições: *generates layer 1 weights* e *generates layer 2 weights* para gerar pesos iniciais aleatórios nestas camadas. O modelo da Figura 14 está didaticamente dividido em quatro partes:
   a) A primeira parte (representada na Figura 14 na cor vermelha), modela a apresentação do banco de dados em épocas. Uma época é a apresentação aleatória de todos os dados uma vez. Para tanto, utilizamos o recurso de temporização no arco da transição *Time*.
   b) A segunda parte (representada na Figura 14 na cor azul), modela a camada de entrada da MLP, inicia-se com a geração aleatória de pesos e limiar, incluindo também a entrada -1 ao limiar. O próximo passo é o somatório dos produtos da entrada pelos

pesos, denominado u e a implementação desta equação no HTCPN-MLP é mostrada a seguir.

```
fun u([ ],v,answervector) =   rev (answervector)
  | u(m,v,answervector) =
  let
       val line = hd m
       fun multiply([ ],[ ],result) = result
       | multiply(line,[ ], result) = result
       | multiply([ ], vector, result) = result
       | multiply(line,vector,result) =
   let
        val el = hd line
        val ev = hd vector
        val x = result + (el*ev)
   in
       multiply(tl line, tl vector, x)
   end
   in
      u(tl m,v,multiply(line,v,0.0)
   ::answervector)
  end
```

c) A terceira parte modela a camada oculta (representada na Figura 14 na cor marrom). A ativação dessa camada é dada pela função tangente hiperbólica, denominada $y_i(t)$ na equação (2.6) e *tghm* no HTCPN-MLP com saídas $[-1,1]$. A implementação desta equação no HTCPN-MLP é mostrada a seguir.

```
fun  tanh ([ ], answervector) = rev (answervector)
   | tanh (v, answervector) =
   let
       val ui = hd v
   in
       tanh (tl v,(1.0-Math.exp(~ ui)) /
            (1.0+Math.exp(~ ui))::answervector)
   end
```

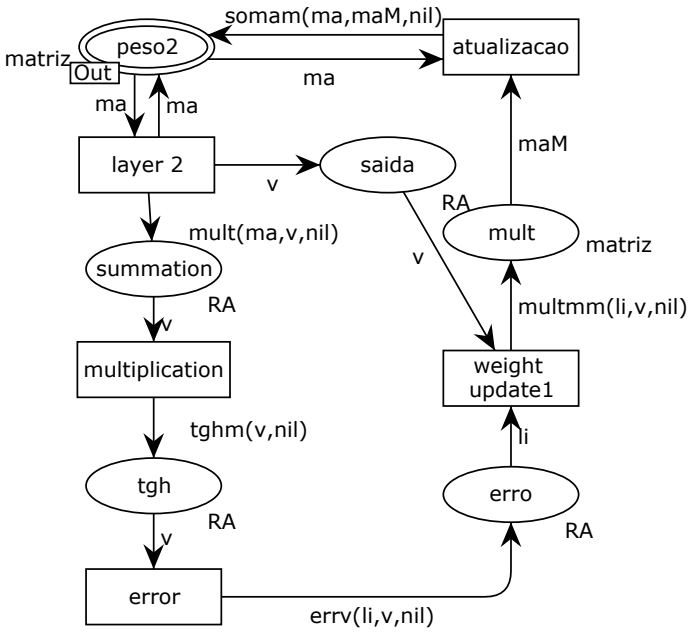
d) A quarta parte (representada na Figura 14 na cor verde): trata-se da camada de saída que contém os mesmos processos da primeira camada e da camada oculta, a saber: geração aleatória de pesos e limiar, cálculo de u_i e a tangente hiperbólica, tendo como saída da rede valores no intervalo $[-1,1]$.

2. Fase de retropropagação dos erros: calculamos o erro na última camada e retropropagamos esse erro para atualizar os pesos e limiar das outras camadas. Essa fase é dividida em três partes:

a) A primeira parte, mostrada na Figura 15, consiste no cálculo de duas funções. A primeira, chamada *gradient* (localizada no arco de saída da transição *error*) calcula a derivada da função de ativação e o gradiente local, a implementação desta equação no HTCPN-MLP é mostrada a seguir.

```
fun gradient([ ],[ ], result:line) = rev(result)
    |gradient(li,[ ], result:line) = rev(result)
    |gradient([ ],v, result:line)  = rev(result)
    |gradient(li,v, result:line)   =
  let
      val el = hd li
      val ev = hd v
      val Ek = el - ev
      val Dk = 0.5*(1.0 - (ev*ev))
  in
      gradient(tl li, tl v, Dk*Ek::result)
  end
```

Figura 15 – Fase de retropropagação do erro durante o treinamento do modelo proposto HTCPN-MLP.



Fonte: Elaborada pelos autores.

b) A segunda parte é a atualização dos pesos e limiares da camada de saída dos neurônios,

em uma forma matricial, esta operação toma a seguinte expressão:

$$\mathbf{W}(t+1) = \mathbf{W}(t) + \alpha\delta(t)\mathbf{y}^T(t), \tag{3.2}$$

em que $\mathbf{W} \in \mathbb{R}^{m\times(q+1)}$ matriz de pesos que conectando os q neurônios ocultos (incluindo limiares) aos m neurônios de saída, $0 < \alpha \ll 1$ é passo de aprendizado, $\delta(t) \in \mathbb{R}^m$ é a coluna de vetores do gradiente local da camada de saída e $\mathbf{y} \in \mathbb{R}^{q+1}$ é um vetor coluna com as saídas dos neurônios ocultos q usados como entrada para a camada de saída (incluindo os limiares).

A implementação desta equação no HTCPN-MLP chamada de `alpha_delta_y` é mostrada a seguir.

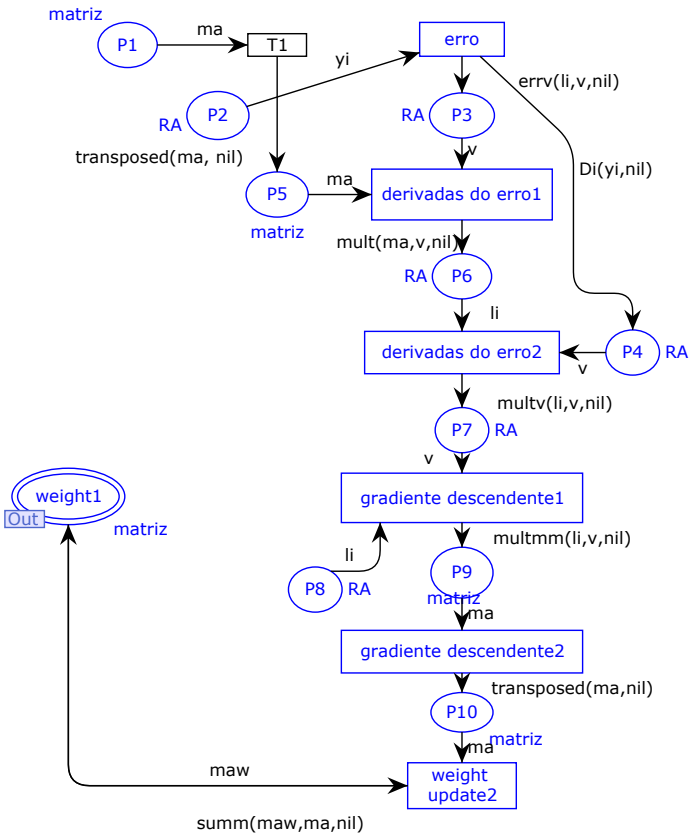
```
fun alpha_delta_y ([ ],[ ],answermatrix:matrix))=
 rev(answermatrix)
    |alpha_delta_y ([ ],v,answermatrix:matrix)) =
 rev(answer_matrix)
    |alpha_delta_y (l,[ ],answermatrix:matrix)) =
 rev(answer_matrix)
    |alpha_delta_y (l,v,answermatrix:matrix)) =
     let
        val line = hd l
         fun a(line:RE, [ ], answervector:line) =
         rev(answervector)|a (line,v,answervector
            )=
     let
         val el = line
         val ev = hd v
     in
       a(line, tl v, (eta* ev * el)::answervector
          )
     end
     in
         alpha_delta_y(tl l, v, a(line,v,nil)::
         answermatrix)
     end
```

c) A terceira parte, mostrada na Figura 16, é a atualização da camada oculta. Temos três estágios para atualizar esta camada:

- O primeiro cálculo é da derivada da tangente logística, chamada `sum_squared_error` (localizada no arco que liga a transição *error* ao lugar *P*4), a implementação desta equação no HTCPN-MLP é mostrada a seguir.

```
fun sum_squared_error([ ], answervector) =
```

Figura 16 – Implementação proposta para a atualização dos pesos dos neurônios da camada oculta.



Fonte: Elaborada pelos autores.

```
rev(answervector)
    |sum_squared_error(yi, answervector) =
  let
     val x = hd yi
  in
     sum_squared_error(tl yi, 0.5*(1.0 -(x*x)
        )
     ::answervector)
  end
```

- A segunda etapa envolve o cálculo dos gradientes locais do j-ésimo neurônio oculto, $\delta_j(t)$, $j = 1,\ldots,q$, usando os gradientes locais dos neurônios de saída propagados de volta através os pesos sinápticos que os conectam ao neurônio oculto j-ésimo. Para isso, usamos a seguinte equação:

$$\delta_j(t) = \left(\sum_{k=1}^{m} w_{kj}(t)\delta_k(t)\right) y'_j(t), \tag{3.3}$$

em que $\delta_k(t)$ é o gradiente local do k-ésimo neurônio da camada oculta, $w_{kj}(t)$ é o peso sináptico conectado ao j-ésimo neurônio oculto com o k-ésimo neurônio de saída e $y'_j(t)$ é a derivada da função de ativação tanh do j-ésimo neurônio oculto.

Esta equação é modelada no HTCPN-MLP pela função `local_gradient` (localizada no arco que liga a transição $T3$ ao lugar $P7$), mostrada a seguir.

```
fun local_gradient([ ],[ ], answervector:line)=
rev(answervector)
   |local_gradient(li,[ ], answervector:line) =
   rev(answervector)
   |local_gradient([ ],v, answervector:line) =
   rev(answervector)
    |local_gradient(li,[ ], answervector:line)=
  let
      val el = hd li
      val ev = hd v
  in
      local_gradient(tl li, tl v, (ev*el)::
         answervector)
  end
```

d) A terceira etapa corresponde à atualização dos pesos e limiares ocultos de acordo com a regra de aprendizado

$$\mathbf{m}_j(t+1) = \mathbf{m}_j(t) + \alpha\delta_j(t)\mathbf{x}(t), \tag{3.4}$$

em que $\mathbf{m}_j(t) \in \mathbb{R}^{p+1}$ é o vetor de pesos do j-ésimo neurônio oculto, $\delta_j(t)$ é o gradiente local calculado na Equação (3.3) e $\mathbf{x}(t)$ é o vetor de entrada atual.

A Equação (3.4) no HTCPN-MLP é implementado com as funções `alpha_delta_y()` (localizada no arco que liga a transição $T4$ ao lugar $P9$),`transposed` (localizada no arco que liga a transição $T5$ ao lugar $P10$) e `sum`(localizada no arco que liga a transição `weight update2` ao lugar `weight1`). segue a seguir

```
fun transposed (nil, mresult) =  rev(mresult)
   |transposed (m, mresult)   =
let
      val li = hd m
      val el = hd li
      val r = li
in
      transpos(tl m, (r::result))
end
fun transp (nil, result:line) = rev(result)
```

```
   |transp (m, result)          =
let
      val li = hd m
      val el = hd li
in
      transp(tl m, (el::result))
end
      val line = transp(m,nil)
      val rest:matrix = transpos(m,nil)
in
      transposed(rest, line::mresult)
end



fun sum ([ ],[ ],answermatrix:matrix:matrix) =
rev(answermatrix)
    |sum ([ ],ma,answermatrix:matrix) =
    rev(answer_matrix)
    |sum (m,[ ],answermatrix:matrix) =
    rev(answer_matrix)
    |sum (m,[ ],answermatrix:matrix) =
let
       val l = hd m
       val ev = hd ma
fun sum_w(l,[ ],answervector):line) = rev(
   answervector))
    |sum_w([ ],v,answervector:line) = rev(
       answervector)
    |sum_w([ ],v,answervector):line) =
in
    sum_w(tl l, tl v, (ev+el)::answervector))
end
in
    sum(tl m, tl ma, sum_w(l,v,nil)::answermatrix
       )
end
```

3.3.2 Validação cruzada da HTCPN-MLP

Uma rede neural artificial visa realizar um mapeamento dos pares entrada-saída de vetores, de modo que a rede aprenda a fornecer uma predição adequada para entradas não vistas durante o treinamento do modelo e e, assim, aproximar com grau de precisão arbitrário a relação de causa e efeito entre as variáveis de entrada e as saídas do problema de interesse. A rede deve, portanto, ter capacidade de generalizar o conhecimento aprendido aplicando-o em novas situações com sucesso.

A validação cruzada é uma técnica estatística bem conhecida que é utilizada no treinamento de RNAs. Durante o treinamento, a capacidade de generalização da rede neural é

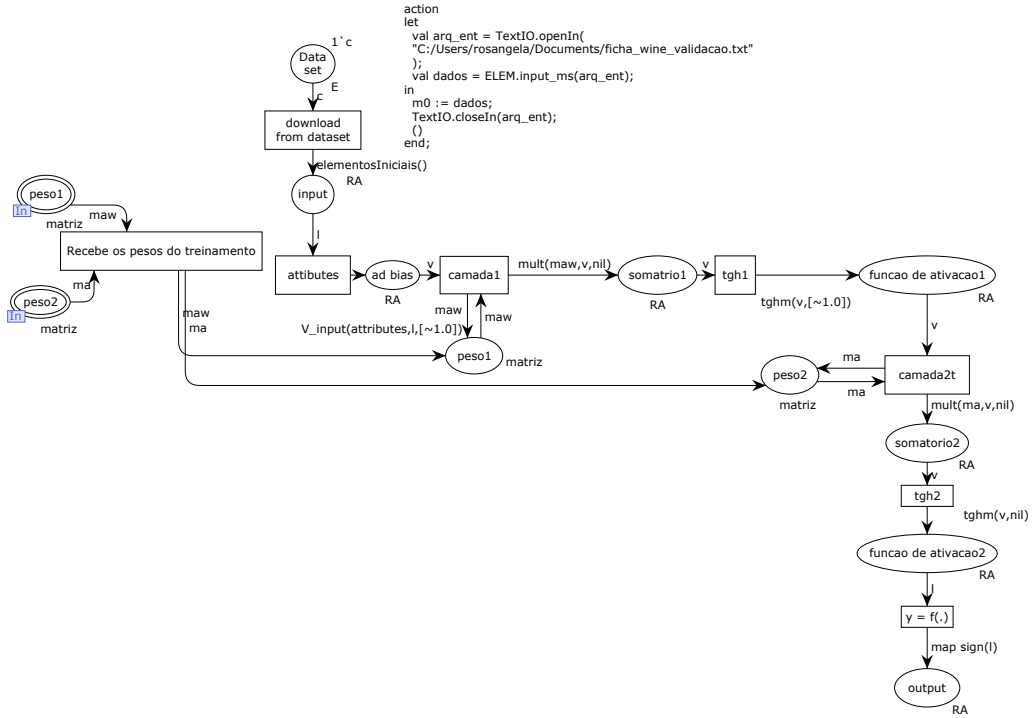
monitorada, estabelecendo de forma sistemática o melhor momento de parada do treinamento para que a rede não tenha um sobreajuste (do inglês *overfitting*). Além desse propósito, a validação cruzada obtém uma estimativa da precisão (ou taxa de erro) do modelo para que seja possível fazer a seleção da melhor arquitetura de uma rede neural. Os dados de treinamento devem ser divididos em dois conjuntos distintos :

- conjunto de treinamento: entre 55% e 90% dos dados são utilizados para atualização de pesos e limiares, nesta tese foi utilizado 60%;
- conjunto de validação: entre 10% e 25% dos dados serão utilizados para verificar a eficiência da rede em relação à sua capacidade de generalização e como critério de parada do treinamento, nesta tese foi utilizado 20%.

Após um número determinado de épocas, o treinamento é interrompido e testamos os pesos e limiares encontrados com o conjunto de validação. Repete-se este procedimento até que a acurácia do conjunto de validação tenha um valor aceitável para o problema.

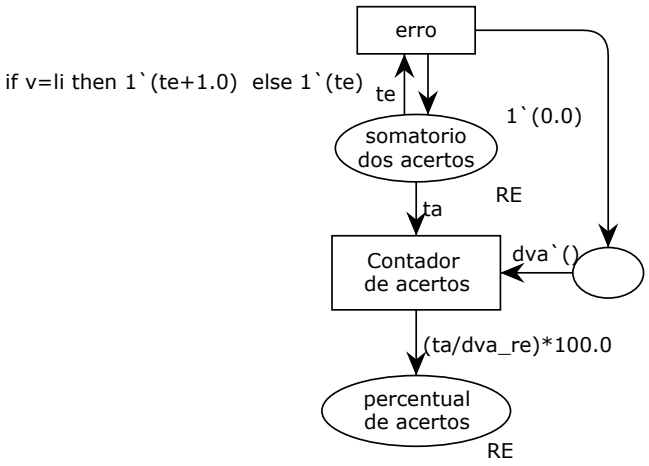
A sub-rede *validation* possui apenas a primeira fase do treinamento, que é a etapa de propagação direta. Nela são comparadas as saídas do HTCPN-MLP com as saídas desejadas e calcula-se a acurácia. Esta sub-rede é mostrada nas Figura 17 e Figura 18.

Figura 17 – Propagação direta através das camadas (ocultas e de saída) durante a fase de validação.



Fonte: Elaborada pelos autores.

Figura 18 – Cálculo da acurácia da HTCPN-MLP na fase validação



Fonte: Elaborada pelos autores.

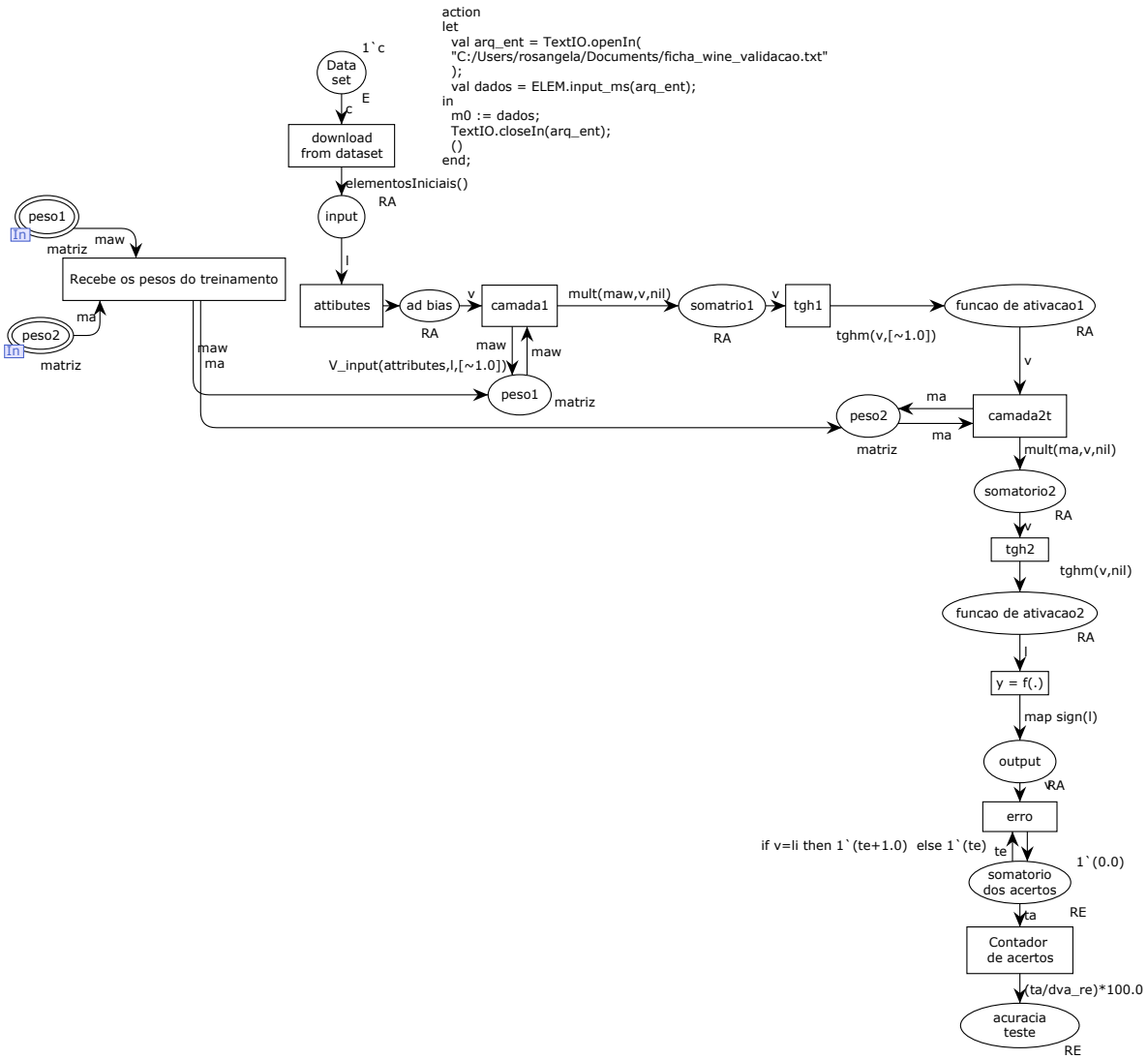
3.3.3 Teste da HTCPN-MLP

O subconjunto teste é composto de 10% a 40% das amostras dos dados, nesta tese foi utilizado 20% para testar se a RNA aprendeu o mapeamento dos pares entrada-saída de maneira satisfatória e tem capacidade de generalização adequada quando fornecido um subconjunto de amostras que não estava no treinamento.

Para testar se o HTCPN-MLP consegue ter uma aprendizagem eficiente, o teste é realizado com 20% dos dados que não foram usados no treinamento para simular condições reais de uso. Esta sub-rede é mostrada na Figura 19 e não será detalhada porque possui as mesmas funções e estrutura da sub-rede *validation* (ver Figura 17 e Figura 18).

Neste capítulo, foi desenvolvido os modelos propostos por esta tese: McCulloch-Pitts-CPN, Perceptron-CPN e HTCPN-MLP, como também a análise das mesmas com alguns banco de dados. No próximo capítulo, será apresentado os bancos de dados e os resultados.

Figura 19 – Sub-rede Teste



Fonte: Elaborada pelos autores.

4 SIMULAÇÕES E ANÁLISE DOS RESULTADOS

Neste capítulo é mostrado os bancos de dados utilizados para testar a HTCPN-MLP, as simulações, os resultados e suas análises.

4.1 Banco de Dados

Para verificar o desempenho do modelo HTCPN-MLP e comparar com os resultados de uma *MLP implementada no Matlab* (MLP-MAT), foram selecionados seis problemas de classificação para *benchmarking* no repositório de aprendizado de máquina da Universidade da Califórnia-Irvine (DUA; GRAFF,), os quais estão resumidos na Tabela 3. Os conjuntos de dados escolhidos abrangem exemplos de diferentes dificuldades e várias dimensões de atributos de entrada, número de classes e cardinalidade do conjunto de dados. Nesta tese, adotamos o sistema operacional Windows 10 e o ambiente de desenvolvimento CPN Tools como plataforma de simulação (CPNTOOLS (2021)).

Tabela 3 – Características de seis conjuntos de dados de classificação utilizados nesta tese.

Banco de dados	Dimensões	Classes	Dados (Treino)	Dados (Teste)
Iris	4	3	120	30
Dermatology	34	6	293	73
Wine	13	3	143	35
WFRN	24	8	4364	1092
Cardiotocography	23	10	1701	425
Seismic-bumps	19	2	2068	516

Fonte: o autor.

Os conjuntos de dados foram pré-processados, de modo que cada atributo tenha média zero e variância unitária antes de serem submetidos às redes. A partição do conjunto de dados foi de 60% para treinamento, 20% para validação e 20% para teste. Os hiperparâmetros dos dois modelos estão indicados na Tabela 4.

Analisamos o modelo HTCPN-MLP quanto à acurácia média (a média de 100 vezes a realização do treinamento, validação e teste) e o erro quadrático médio do treinamento.

Nas seções a seguir apresentamos os resultados dos seguintes bancos de dados: Iris, Dermatology, Wine, Wall-Following Robot, Cardiotocography e Seismic-bumps.

Tabela 4 – Hiperparâmetros dos modelos avaliados.

Banco de dados	Número de neurônios ocultos	Taxa de aprendizagem	Número de épocas
	HTCPN \ MAT	HTCPN \ MAT	HTCPN \ MAT
Iris	3	0,5	100
Dermatology	10	0,5	100
Wine	3	0,5	100
Wall-Following Robot	8	0,1	100
Cardiotocography	7	0,1	100
Seismic-bumps	2	0,1	100

Fonte: o autor.

4.2 O Conjunto de Dados Iris

O conjunto de dados Iris [1] é um dos mais conhecidos na área de aprendizado de máquina. Para a classificação, foram utilizadas quatro variáveis morfológicas: comprimento e largura das sépalas e das pétalas, em centímetros. Esse conjunto de dados contém 150 amostras igualmente divididas em três classes, e foi utilizado para testar o modelo perceptron-CPN considerando apenas suas duas classes linearmente separáveis. Com o modelo HTCPN-MLP vamos testá-lo com suas três classes, sendo assim um problema não linearmente separável.

O modelo proposto HTCPN-MLP e a implementação em Matlab foram treinados com uma camada oculta e com um número crescente de neurônios ocultos até atingir três neurônios. Foram executadas 100 épocas de treinamento e teste, com a topologia descrita acima, e o conjunto de treinamento e teste foi dividido aleatoriamente, atingindo uma acurácia média de 98% no conjunto de teste. À medida que as redes foram treinadas, o erro médio quadrático foi diminuindo produzindo a curva de aprendizado mostrada no gráfico da Figura 20.

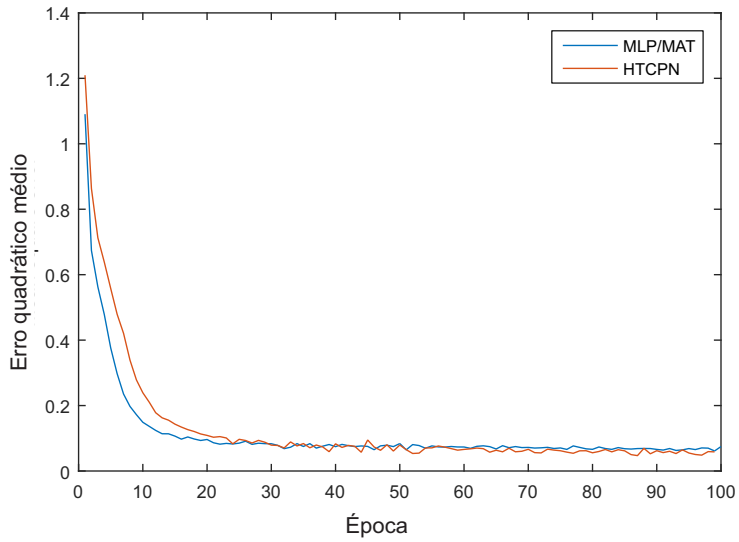
4.3 O Conjunto de Dados Wine

Outro conjunto de dados bem conhecido nessa área é o Wine [2], que consiste em um problema de classificar a origem do vinho de três culturas diferentes em uma região da Itália a partir de análises químicas. Composto por 178 padrões, divididos em 59 amostras da primeira safra, 71 da segunda e 48 da terceira. A análise química é composta por 13 atributos.

O modelo proposto HTCPN-MLP e a implementação em Matlab foram treinados com a mesma topologia: uma única camada oculta com três neurônios. Após 100 épocas de treinamento e teste, obtém-se uma acurácia média de 94% no teste, tanto no MLP-MAT quanto

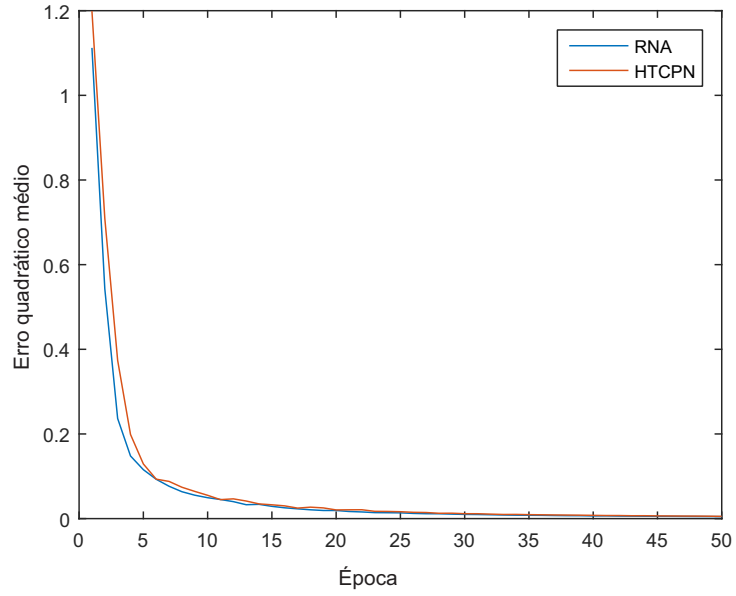
[1] https://archive.ics.uci.edu/ml/datasets/Iris
[2] https://archive.ics.uci.edu/ml/datasets/Wine

Figura 20 – Curvas de aprendizado dos modelos HTCPN-MLP e a MLP/Matlab para o conjunto Iris.



Fonte: Elaborada pelos autores.

Figura 21 – Curvas de aprendizado dos modelos HTCPN-MLP e a MLP/Matlab para o conjunto Wine.



Fonte: Elaborada pelos autores.

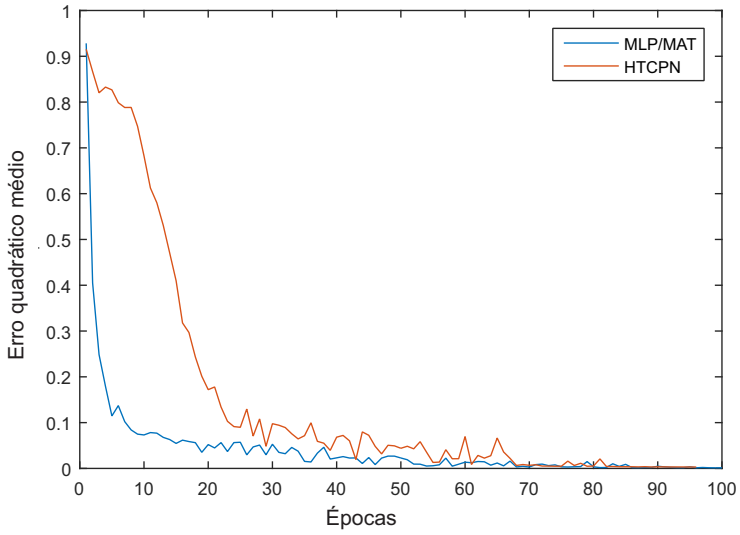
no modelo HTCPN-MLP. O gráfico da curva de aprendizado é mostrado na Figura 21. O erro quadrático médio diminui durante o treinamento até atingir um valor próximo a zero.

4.4 O Conjunto de dados Dermatology

O próximo conjunto de dados é o Dermatology [3] que consiste em classificar seis doenças de pele: psoríase, dermatite sebórica, líquen plano, pitiríase rósea, dermatite crônica e pitiríase rubra pilar; com os seguintes números de amostra, respectivamente: 112, 61, 72, 49, 52 e 20, perfazendo um total de 366 amostras, cada uma com 34 atributos, divididos em 12 clínicos e 22 histopatológicos.

Tanto a implementação no matlab quanto o modelo HTCPN-MLP foram treinados com uma camada oculta composta por dez neurônios. Foram realizadas cem épocas de treinamento e teste, com a topologia descrita acima, e o conjunto de treinamento e teste foi dividido aleatoriamente, atingindo uma acurácia média de 96% no conjunto de testes para ambos os modelos avaliados. O gráfico da curva de aprendizado é mostrado na Figura 22, nas 100 épocas de treinamento, o EQM está diminuindo até próximo de zero.

Figura 22 – Curvas de aprendizado dos modelos HTCPN-MLP e a MLP/Matlab para o conjunto Dermatology.



Fonte: Elaborada pelos autores.

Na Tabela 5 são mostrados os resultados obtidos pelos três conjuntos de dados: *Iris, Wine e Dermatology*. Pode-se observar que não há diferenças significativas entre os resultados da MLP-MAT e do HTCPN-MLP.

[3] https://archive.ics.uci.edu/ml/datasets/Dermatology

Tabela 5 – Tabela resumo com as medidas de desempenho das redes avaliadas para os conjuntos de dados: *Iris, Wine e Dermatology*

Estatísticas de Desempenho	HTCPN-MLP			MLP-MAT		
	Iris	*Dermatology*	*Wine*	*Iris*	*Dermatology*	*Wine*
EQM	0,0011	0,0013	0,0017	0,0021	0,0013	0,0017
acuracia média(%)	98,0	91,0	97,3	96,8	96,6	97,5
Min(%)	91,0	80,0	91,6	90,0	89,0	91,7
Máx(%)	100,0	98,0	100,0	100,0	100,0	100,0
Mediana	97,0	96,6	96,1	96,7	97,2	97,2
Desvio padrão	2,7	1,8	2,4	2,7	1,7	2,3

Fonte: o autor.

4.5 Wall-Following Robot Navigation

O conjunto de dados *Wall-Following Robot Navigation* (WFRN), consiste em 24 sensores igualmente espaçados ao longo da circunferência do robô SCITOS-G5: 12 sensores na frente e 12 atrás e o objetivo é aprender a comandar o robô de maneira que ele realize um caminho sem colisões. O conjunto de dados contém os valores brutos de medição de todos os 24 sensores de ultrassom e o rótulo da classe correspondente. As leituras do sensor são apresentadas a uma taxa de 9 amostras por segundo, perfazendo um total de 5456 amostras.

O modelo proposto HTCPN-MLP e a MLP-MAT foram treinados com uma camada oculta composta por oito neurônios. Foram realizadas 100 épocas de treinamento e teste, com a topologia descrita acima, e o conjunto de treinamento e teste foi dividido aleatoriamente, atingindo uma acurácia média de 80% no conjunto de testes. O gráfico da curva de aprendizado é mostrado na Figura 23, a partir da qual percebe-se que o erro quadrático médio diminui durante o treinamento até atingir um valor próximo de zero.

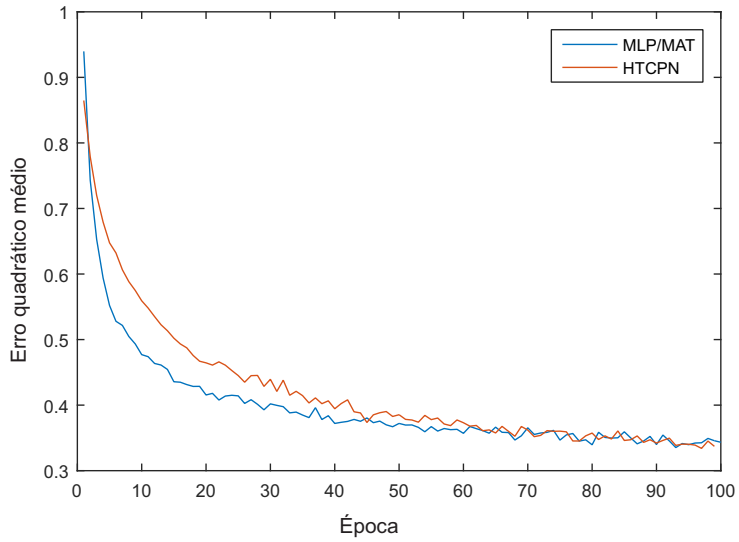
4.6 O conjunto de dados Seismic-Bumps [4]

Este conjunto é um problema de classificação binária composto por dois estados que são interpretados como “perigoso” e “não perigoso” em uma mina de carvão, referindo-se a um problema de projeção sísmica de alta energia (maior que 10^4 J). O objetivo deste conjunto de dados é prever (com precisão em relação à hora e data) o aumento da atividade sísmica que pode causar uma explosão de rocha. O conjunto de dados contém 2584 padrões e 19 atributos.

O desafio deste conjunto de dados consiste na distribuição desequilibrada de positivo (estado perigoso) que são apenas 170 amostras e negativo (estado não perigoso) que são 2.414

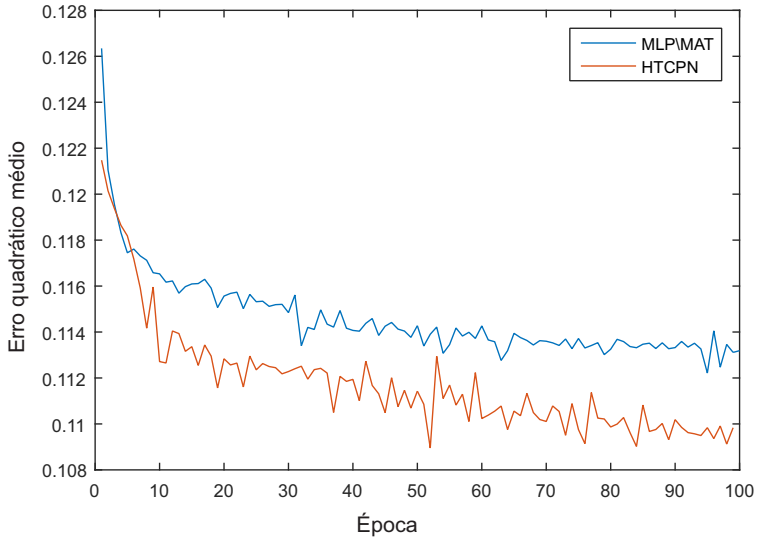
[4] https://archive.ics.uci.edu/ml/datasets/seismic-bumps

Figura 23 – Curvas de aprendizado dos modelos HTCPN-MLP e a MLP/Matlab para o conjunto WFRN



Fonte: Elaborada pelos autores.

Figura 24 – Curvas de aprendizado dos modelos HTCPN-MLP e a MLP/Matlab para o conjunto *Seismic-Bumps.*

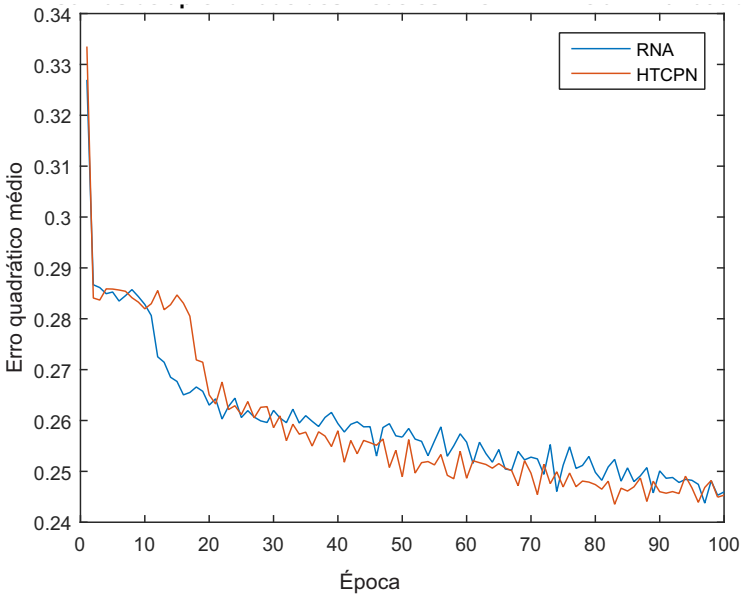


Fonte: Elaborada pelos autores.

amostras. Dado esse desequilíbrio, a partição desse conjunto de dados foi de 80% de cada classe para treinamento e 20% de cada classe para teste. Chegamos à arquitetura de uma camada oculta com dois neurônios pelo método do número crescente. Após 100 épocas, atingimos uma acurácia média de 93% no conjunto de testes. O gráfico da curva de aprendizado é mostrado na Figura 24. O erro quadrático médio diminui durante o treinamento de 100 épocas até atingir um valor de 0.11.

Na Tabela 6 são mostrados os resultados obtidos pelos três conjuntos de dados:

Figura 25 – Curvas de aprendizado dos modelos HTCPN-MLP e a MLP/Matlab para o conjunto *Cardiotography*



Fonte: Elaborada pelos autores.

navegação de robô que segue a parede (WFRN), cardiotocografia (cardi) e colisões sísmicas (seismic-bumps). Observe que tanto estes conjuntos de dados, quanto os conjuntos de dados da Tabela 5 o HTCPN-MLP e a MLP-MAT possuem resultados semelhantes.

4.7 O conjunto de dados Cardiotocography [5]

Este conjunto de dados consiste em medições da frequência cardíaca fetal (FCF) e da contração uterina em cardiotocografia. Contém 2.126 amostras e 10 classes cujos rótulos estão relacionados ao padrão morfológico da FCF.

O modelo proposto HTCPN-MLP e o MLP-MAT foram treinados com sete neurônios ocultos com a mesma metodologia de avaliação dos experimentos anteriores. Uma acurácia média de 75% foi obtida no teste por ambos os modelos avaliados. As curvas de aprendizado típicas baseadas em EQM para este experimento são mostradas na Fig. 25.

Neste capítulo, foi apresentado os bancos de dados e os resultados obtidos nas tabelas 5 e 6, pode-se perceber que os resultados são muito próximos, não existe diferença significativa do HTCPN-MLP e MLP-MAT. No próximo capítulo, será comentado o desenvolvimento de outro modelo proposto por esta tese, redes de Petri coloridas isomórficas a redes neurais construtivas.

[5] https://archive.ics.uci.edu/ml/datasets/Cardiotocography

Tabela 6 – Tabela resumo com as medidas de desempenho das redes avaliadas para os conjuntos de dados WFRN, Cardiotocography (Cardi) e Seismic-Bumps (SB) data sets.

Estatísticas de desempenho	HTCPN-MLP			MLP-MAT		
	WFRN	*Cardi*	*SB*	*WFRN*	*Cardi*	SB
EQM	0,10	0,37	0,108	0,39	0,34	0,111
Acuracia média(%)	90,8	85,0	93,6	87,1	86,7	97,5
Min(%)	80,8	83,0	93,0	80,0	83,0	94,0
Max(%)	91,8	100,0	94,0	90,0	100,0	100,0
Mediana	86,9	87,0	93,5	85,0	86,8	95,0
Desvio padrão	2,0	1,8	0,8	2,3	1,7	1,6

Fonte: o autor.

5 REDES DE PETRI COLORIDAS PARA ANÁLISE E MODELAGEM DAS REDES NEURAIS CONSTRUTIVAS

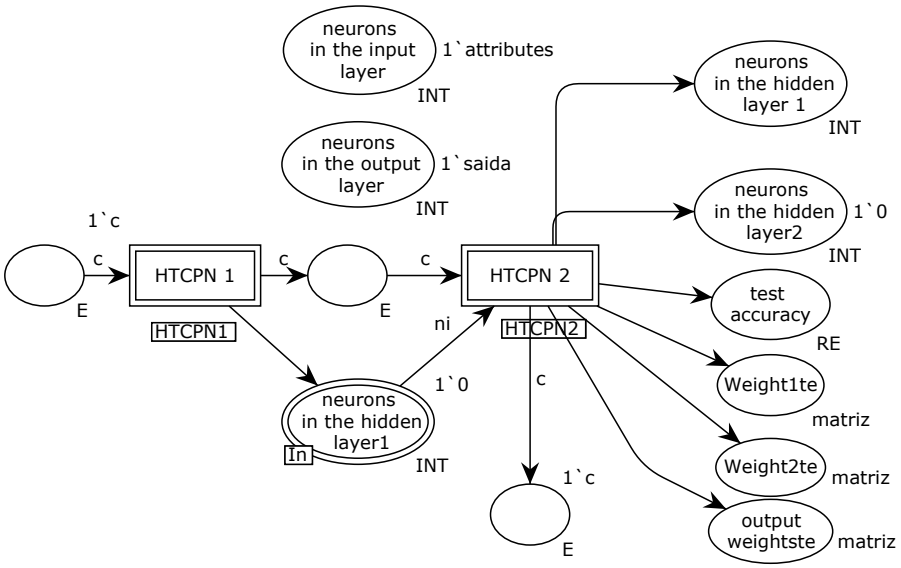
O processo de aprendizado das redes neurais pode enfrentar problemas de *underfitting* e *overfitting*. Redes neurais com arquitetura muito pequenas, muitas vezes não são capazes de extrair relacionamentos complexos do conjunto de treinamento, conduzindo ao problema de *underfitting*. Em contra partida, redes neurais com arquitetura grande pode extrair relacionamentos muito específicos, até mesmo ruídos, tornando a rede tendenciosa aos padrões de treinamento, o que nos leva ao *overfitting*. A escolha de uma arquitetura otimizada através de uma abordagem empírica tem estimulado outras alternativas mais eficientes, temos dois principais métodos: a poda (REED, 1993) e as redes construtivas. A poda, cuja idéia é iniciar com uma arquitetura de dimensão elevada e ir retirando neurônios ou conexões até que se obtenha uma arquitetura adequada.

Contudo, existe dificuldade de determinação da arquitetura inicial que contenha uma estrutura necessária para encontrar a solução do problema. Devido a isso, geralmente escolhe-se arquiteturas fortemente sobredimensionadas. Sendo assim, a maior parte do treinamento será realizado com a rede sobredimensionada, aumentando significativamente o custo computacional. Outro problema que a poda enfrenta é de geralmente escolher a arquitetura com maior dimensão, pois a retirada de neurônio ou conexões param, quando se encontra uma solução aceitável para o problema e inúmeras redes neurais de diferentes dimensões são capazes de representar soluções aceitáveis. Além disso, vários neurônios são retirados simultaneamente, com isso não temos uma estimativa confiável do efeito da poda que cada neurônio pode causar junto ao erro de aproximação.

Outro método de resolver o problema da otimização de arquiteturas são as redes neurais construtivas. Estas redes conseguem aprender a resolver o problema de interesse e ao mesmo tempo criar sua própria topologia.

As redes construtivas são iniciadas com uma estrutura mínima, novos neurônios são adicionados um a um, sempre que necessário, viabilizando a construção dinâmica de multiplas camadas enquanto realiza o seu treinamento. Uma rede construtiva chamada de *cascade-correlation* foi proposta por Scott E. Fahlman e Christian Lebiere (FAHLMAN; LEBIERE, 1990) e combina três idéias: arquitetura em cascata (neurônios são adicionadas um a um e congelados), algoritmo de aprendizagem *quickprop* (FAHLMAN *et al.*, 1988) e aprendizado profundo (essa rede é considerada uma das primeiras abordagens ao que hoje se conveciou chamar de aprendizado profundo). O algoritmo *quickprop* é mais rápido do que o de retropropagação,

Figura 26 – Estrutura do modelo construtivo proposto CA-CPN.



Fonte: Elaborada pelos autores.

porém não é eficiente em alguns domínios. isto se deve ao fato que a aproximação realizada pelo algoritmo, da derivada de segunda ordem da função de perda, com um quociente de diferença simples, baseia-se na suposição de que essa derivada de segunda ordem é uma função contínua, porém quando a derivada de segunda ordem é descontínua, o comportamento do algoritmo pode se tornar não confiável. Um estudo mais detalhado encontra-se em)brust2016neither.

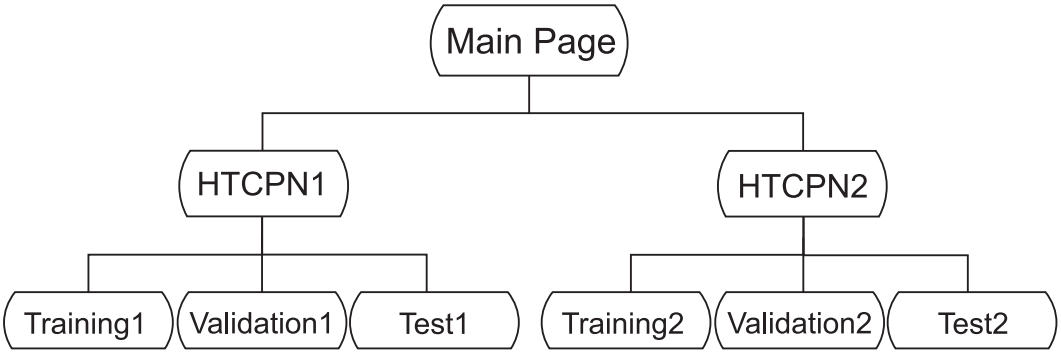
Neste trabalho, implementamos uma rede de Petri colorida temporizada para projetar uma rede MLP com algoritmo construtivo, chamada de *constructive algorithms for colored Petri net* (CA-CPN), ilustrada na Figura 26. O modelo CA-CPN começa com uma arquitetura de RNA mínima, podendo evoluir para arquiteturas mais complexas a cada iteração. A cada iteração acrescenta-se um neurônio e a rede é treinada com o algoritmo de retropropagação do erro. Escolhemos este algoritmo de aprendizado pela sua ampla utilização no treinamento de redes neurais multicamadas.

5.1 Estrutura do Modelo CA-CPN

O modelo CA-CPN é uma rede hierarquica que contém uma página principal *Main page*, duas sub-redes chamadas de *HTCPN1* e *HTCPN2* e cada sub-rede contém três sub-redes chamadas de *training*, *validation* e *test*; um modelo mais abstrato é mostrado na Figura 27.

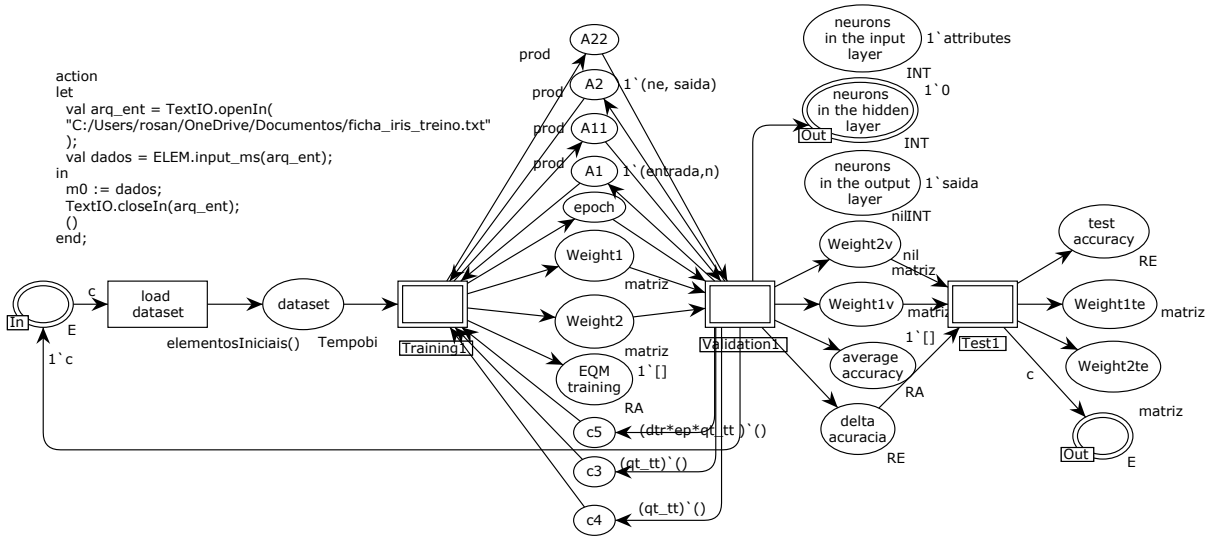
O modelo mostrado na Figura 26 trata-se da pagina principal que contém duas transições hierárquicas: *HTCPN1* e *HTCPN2* . Inicia-se com o modelo HTCPN1 que tem uma MLP, composta por uma camada de entrada com p neurônios, uma camada oculta com

Figura 27 – Diagrama abstrato do modelo construtivo proposto CA-CPN.



Fonte: Elaborada pelos autores.

Figura 28 – Estrutura do modelo HTCPN1

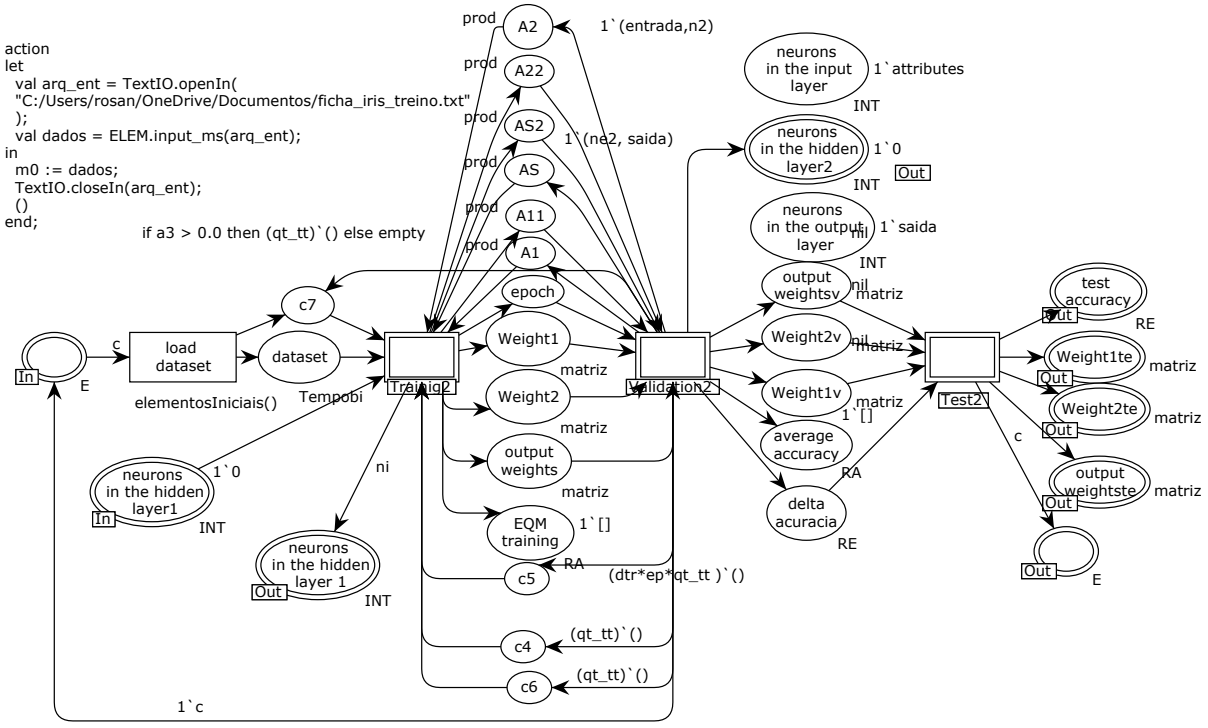


Fonte: Elaborada pelos autores.

apenas um neurônio e uma camada de saída com *m* neurônios. A constante *p* é a quantidade de atributos do banco de dados e *m* depende da forma como as saídas foram codificadas para serem resolvidas por uma rede neural. O modelo HTCPN1, mostrada na Figura 28 otimiza a quantidade de neurônios na primeira camada oculta, dando como resultado na pagina principal, essa quantidade de neurônios. Se a rede alcançar *n* neurônios na primeira camada oculta (nesta tese utilizamos *n = 10*), acrescentamos a segunda camada oculta e inicializamos o modelo HTCPN2, mostrada na Figura 29, que recebe a quantidade de neurônios da primeira camada oculta e otimiza a quantidade de neurônios na segunda camada oculta.

A otimização da quantidade de neurônios no HTCPN é realizada de forma crescente, de modo que inicia-se com uma topologia mínima (ver Figura 30), apenas um neurônio na camada oculta, e a cada iteração do algoritmo, sendo uma iteração a média de 100 épocas de treinamento e teste com a mesma arquitetura, acrescenta-se um neurônio (ver Figura 31) até que o aumento de neurônios nesta camada não melhora a acurácia média ou se chegue a um

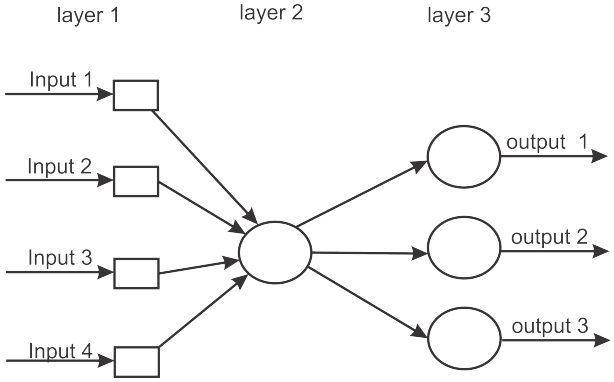
Figura 29 – Estrutura do modelo HTCPN2



Fonte: Elaborada pelos autores.

número máximo permitido de neurônios na camada. Para finalizar, o algoritmo escolhe a menor quantidade de neurônios com a melhor acurácia média, resultando assim em uma arquitetura otimizada.

Figura 30 – Arquitetura do HTCPN na primeira iteração

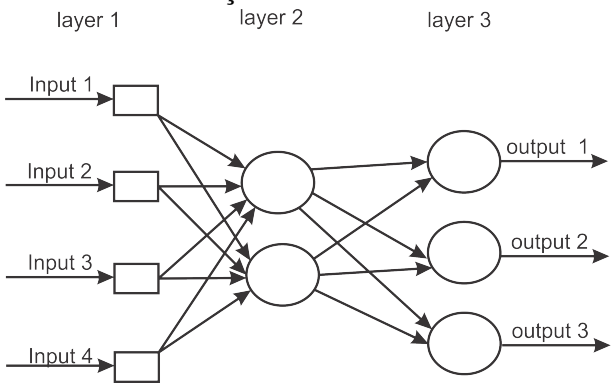


Fonte: Elaborada pelos autores.

5.2 Estrutura do Modelo HTCPN1

O modelo HTCPN1 é uma rede hierárquica composta de três sub-redes: *training*, *validation* e *test*. A sub-rede *Training*, mostrada na Figura 14 recebe o banco de dados e gera uma MLP com uma camada de entrada com p neurônios, uma camada oculta com apenas um

Figura 31 – Arquitetura do HTCPN na segunda iteração



Fonte: Elaborada pelos autores.

neurônio e uma camada de saída com com *m* neurônios. Os pesos iniciais são gerados de forma aleatória e realiza-se o treinamento. Os pesos são atualizados no treinamento e são validados na sub-rede *validation*, mostrada nas Figuras 17 e 18. Realiza-se o treinamento e validação 100 vezes e calcula-se a acurácia média desta arquitetura. Repete-se o procedimento anterior, acrescentando um neurônio na camada oculta. Se a acurácia média estiver aumentando, o modelo HTCPN1 continua acrescentando neurônios na camada oculta, até que a acurácia média não se modifique ou até chegar a acurácia média desejada ou ou se chegue a n neurônios. Após esse procedimento, iniciamos o teste na sub-rede *Test*, mostrada na Figura 19 enviando para a rede principal a arquitetura otimizada.

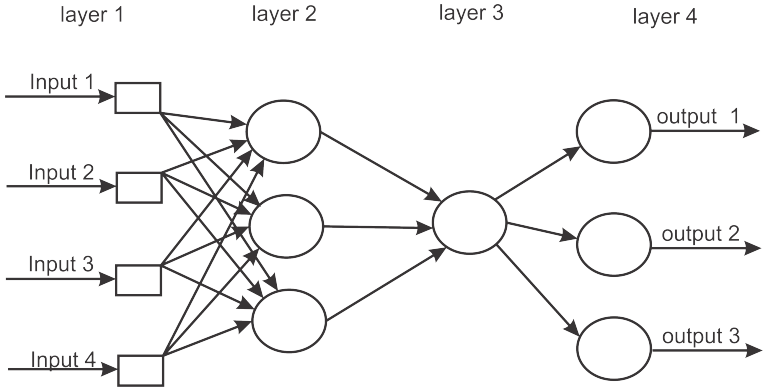
5.3 Estrutura do modelo HTCPN2

Se a acurácia média desejada não for obtida com uma camada oculta, a CA-CPN inicia a sub-rede HTCPN2, mostrada na Figura 29 que gera uma MLP com duas camadas ocultas, composta por *p* neurônios na primeira camada, *q* neurônios na primeira camada oculta, um neurônio na segunda camada oculta e *m* neurônios na camada de saída (ver Figura 32), sendo *p* e *m*, respectivamente, a quantidade de atributos e número de classes e *q* a quantidade de neurônios encontrados para o modelo HTCPN1. São acrescentados neurônios na segunda camada oculta até encontrar uma arquitetura otimizada seguindo o mesmo algoritmo do modelo HTCPN1. A Figura 33 ilustra a segunda iteração do modelo HTCPN2.

5.4 Simulação e Discussão

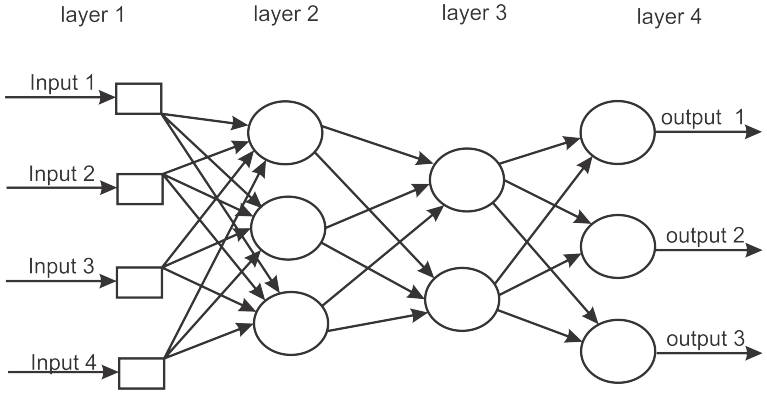
Para verificar o desempenho do modelo CA-CPN e comparar os resultados com o HTCPN-MLP e com a MLP implantada no Matlab, foram selecionados três problemas de

Figura 32 – Arquitetura do modelo HTCPN2 na primeira iteração da MLP com 2 camadas ocultas.



Fonte: Elaborada pelos autores.

Figura 33 – Arquitetura do modelo HTCPN1 na segunda iteração da MLP com 2 camadas ocultas.



Fonte: Elaborada pelos autores.

classificação: iris, *dermatology* e *wine* que já foram testados no HTCPN-MLP.

Os conjuntos de dados foram pré-processados, de modo que cada atributo tenha média zero e variância unitária antes de serem submetidos às redes. A partição do conjunto de dados foi de 60% para treinamento, 20% para validação e 20% para teste.

Nas Tabelas 7, 8 e 9 são mostrados os resultados obtidos pelos três conjuntos de dados. Pode-se observar que não há diferenças significativas entre os resultados da CA-CPN e do HTCPN-MLP. Caso o usuário queira apenas saber a melhor arquitetura, uma boa métrica é a moda do número de neurônios em dez rodadas, treina várias vezes e vê qual o número de neurônios ocultos resultantes. Percebe-se que a moda do banco de dados Iris é 3, do Wine é 3 e do Dermatology é 9 ou 10. Para testar o modelo HTCPN2, foi realizado outro teste com o conjunto de dados *Dermatology*, nesse teste, aumentou-se a acurácia desejada, o modelo CA-CPN tentando chegar nessa acurácia, aumenta a quantidade de camadas, exigindo a inserção da segunda camada oculta. Os resultados são mostrados na Tabela 10. Na segunda rodada, a primeira camada oculta tem 10 neurônios e a segunda camada oculta tem 2 neurônios; na terceira

rodada, a primeira camada oculta tem 10 neurônios e a segunda camada oculta tem 1 neurônio. O mesmo acontece nas rodadas: 2, 3, 4, 5, 8, 9 e 10.

Tabela 7 – Medidas de desempenho do modelo CA-CPN para o conjunto de dados Iris

Grandeza Observada	Rodadas									
	1ª	2ª	3ª	4ª	5ª	6ª	7ª	8ª	9ª	10ª
nº de neurônios	3	2	2	3	4	2	3	3	3	4
acuracia média(%)	98,0	97,8	96,6	97,8	98,9	96,6	96,6	97,8	98,9	98,9

Fonte: o autor.

Tabela 8 – Medidas de desempenho do modelo CA-CPN para o conjunto de dados *Wine*

Grandeza Observada	Rodadas									
	1ª	2ª	3ª	4ª	5ª	6ª	7ª	8ª	9ª	10ª
nº de neurônios	2	2	3	2	3	3	2	3	4	3
acuracia média(%)	95,6	97,1	96,6	95,4	98,1	97,3	96,6	97,6	98,1	97,1

Fonte: o autor.

Tabela 9 – Medidas de desempenho do modelo CA-CPN para o conjunto de dados *Dermatology*

Grandeza Observada	Rodadas									
	1ª	2ª	3ª	4ª	5ª	6ª	7ª	8ª	9ª	10ª
nº de neurônios	7	8	10	9	8	10	10	9	7	9
acuracia média(%)	89,8	90,1	92,3	94,1	92,3	97,6	96,6	95,3	90,6	96,8

Fonte: o autor.

Tabela 10 – Medidas de desempenho do modelo CA-CPN para o conjunto de dados *Dermatology*

Grandeza Observada	Rodadas									
	1ª	2ª	3ª	4ª	5ª	6ª	7ª	8ª	9ª	10ª
nº de neurônios	10	10-2	10-1	10-2	10-1	10	10	10-2	10-2	10-2
acuracia média(%)	94	93	94,2	94,1	95,3	94,2	95,1	95,3	96,8	97,8

Fonte: o autor.

Neste capítulo, foi apresentado o modelo de redes de Petri coloridas isomórficas as redes neurais construtivas e seus resultados com três banco de dados e no próximo capítulo será realizado a conclusão desta tese.

6 CONCLUSÕES E TRABALHOS FUTUROS

Nesta tese introduzimos uma nova abordagem para a construção de arquiteturas de redes neurais multicamadas totalmente adaptáveis que podem ser treinadas pelo algoritmo de retropropagação usando o arcabouço das redes de Petri coloridas hierárquicas e temporizadas. O modelo proposto foi denominado HTCPN-MLP e foi construído usando modelos intermediários de redes neurais baseadas em PN desenvolvidos para servir como blocos de construção para o HTCPN-MLP mais complexo, tais como o modelo perceptron-CPN e o modelo McCulloch-Pitts-CPN de um neurônio.

A abordagem proposta tem as seguintes características:

1. capacidade de carregar qualquer conjunto de dados;
2. capacidade de gerar pesos aleatórios para inicialização do treinamento;
3. uso de restrições de tempo para apresentar dados em épocas;
4. estrutura hierárquica e modelagem de funções de ativação;
5. implementação baseada em CPNs enquanto armazena informações em seus tokens;
6. capacidade de propagar o erro nos moldes do algoritmo de retropropagação;
7. todas as etapas de aprendizagem envolve treinamento, teste e validação..

O modelo CA-CPN tem as mesma características supracitadas, além de ser uma importante ferramenta de determinação automática de arquiteturas de redes neurais, com as seguintes características:

1. inicia-se com uma estrutura mínima, que vai sendo incrementada com o passar das iterações;
2. Possui um custo computacional significativamente menor do que a poda, pois a maior parte do processo de aproximação da poda é realizado considerando-se redes neurais sobredimensionadas;
3. Como geralmente muitas redes neurais de diferentes arquiteturas são capazes de representar soluções aceitáveis para o problema, o modelo CA-CPN escolhe a menor arquitetura, enquanto as técnicas de poda em geral finalizam com uma arquiteura com maior número de neurônios ocultos;
4. Facilidade na especificação da arquitetura inicial, a saber: uma camada oculta com um neurônio. Enquanto a poda é nem sempre é fácil determinar uma arquitetura inicial;
5. Automotização do processo de determinação da arquiteura ótima da rede neural, bastando iniciá-la e deixá-la evoluir com o passar do treinamento.

O estudo sistemático realizado que levou ao desenvolvimento dos modelos HTCN-MLP e CA-CPN, também deixou algumas questões a serem tratadas em estudos futuros. Algumas delas são listadas a seguir.

1. Desenvolver extensões dos modelos propostos nesta tese em problemas de regressão. Os modelos propostos foram testados apenas em problemas de classificação, em um futuro próximo, pretende-se testar conjuntos de dados regressão;
2. Extensão para outras redes neurais, como redes randomizadas, por exemplo: ELM (HUANG *et al.*, 2006) e RVFL (PAO *et al.*, 1994). São redes neurais alternativas à rede MLP de uma camada oculta, com um aprendizado rápido.
3. O modelo HTCPN-MLP tem um processamento mais rápido que o do modelo MLP-MAT. Pretende-se realizar um amplo estudo sobre o custo computacional dos diversos modelos avaliados nesta tese, bem como aqueles que surgirão de suas extensões baseadas em redes neurais randomizadas.

REFERÊNCIAS

AHSON, S. I. Petri net models of fuzzy neural networks. **IEEE Transactions on systems, man, and cybernetics**, IEEE, v. 25, n. 6, p. 926–932, 1995.

ARIVUDAINAMBI, D.; KA, V. K.; VISU, P. *et al.* Malware traffic classification using principal component analysis and artificial neural network for extreme surveillance. **Computer Communications**, Elsevier, v. 147, p. 50–57, 2019.

BOUHOUNE, K.; YAZID, K.; BOUCHERIT, M. S.; CHÉRITI, A. Hybrid control of the three phase induction machine using artificial neural networks and fuzzy logic. **Applied Soft Computing**, Elsevier, v. 55, p. 289–301, 2017.

CABANEROS, S. M.; CALAUTIT, J. K.; HUGHES, B. R. A review of artificial neural network models for ambient air pollution prediction. **Environmental Modelling & Software**, Elsevier, v. 119, p. 285–304, 2019.

CHAN, S. M.; KE, J.-S.; CHANG, J. F. Knowledge representation using fuzzy Petri net. **IEEE Transactions on Knowledge and Data Engineering**, IEEE, v. 2, n. 3, p. 311–319, 1990.

CHOW, T. W.; LI, J.-Y. Higher-order petri net models based on artificial neural networks. **Artificial Intelligence**, Elsevier, v. 92, n. 1-2, p. 289–300, 1997.

CPNTOOLS. **CPNTOOLS**. 2021. Disponível em: <https://cpntools.org/>. Acesso em: 24 abril. 2021.

CUN, Y. L. Learning process in an asymmetric threshold network. In: BIENENSTOCK, E.; SOULIÉ, F. F.; WEISBUCH, G. (Ed.). **Disordered Systems and Biological Organization**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986. p. 233–240. ISBN 978-3-642-82657-3.

DUA, D.; GRAFF, C. **UCI Machine Learning Repository**. Disponível em: <http://archive.ics.uci.edu/ml>. Acesso em: 28 jun. 2016.

FAHLMAN, S. E.; LEBIERE, C. **The cascade-correlation learning architecture**. [S.l.], 1990.

FAHLMAN, S. E. *et al.* **An empirical study of learning speed in back-propagation networks**. [S.l.]: Carnegie Mellon University, Computer Science Department Pittsburgh, PA, USA, 1988.

GARG, M. L.; AHSON, S. I.; GUPTA, P. V. A fuzzy Petri net for knowledge representation and reasoning. **Information Processing Letters**, Elsevier, v. 39, n. 3, p. 165–171, 1991.

GENRICH, H. J.; THIELER-MEVISSEN, G. The calculus of facts. In: **Lecture Notes in Computer Science**. [S.l.]: Springer-Verlag, 1976. v. 45, p. 588–595.

HABIB, M. K.; NEWCOMB, R. W. Neuron type processor modeling using a timed Petri net. **IEEE Transactions on Neural Networks**, IEEE, v. 1, n. 4, p. 282–289, 1990.

HABIB, M. K.; NEWCOMB, R. W. Neuron type processor modeling using a timed petri net. **IEEE transactions on neural networks**, IEEE, v. 1, n. 4, p. 282–289, 1990.

HUANG, G.-B.; ZHU, Q.-Y.; SIEW, C.-K. Extreme learning machine: theory and applications. **Neurocomputing**, Elsevier, v. 70, n. 1-3, p. 489–501, 2006.

JENSEN, K.; KRISTENSEN, L. M. **Coloured Petri nets: modelling and validation of concurrent systems**. [S.l.]: Springer Science & Business Media, 2009.

KORIEM, S. M. CN-nets for modeling and analyzing neural networks. **Artificial Intelligence**, Elsevier, v. 13, p. 19–47, 2001.

KORIEM, S. M. Cn-nets for modeling and analyzing neural networks. **Journal of King Saud University-Computer and Information Sciences**, Elsevier, v. 13, p. 21–49, 2001.

LI, L.; XIE, Y.; CEN, L.; ZENG, Z. A novel cause analysis approach of grey reasoning petri net based on matrix operations. **Applied Intelligence**, Springer, v. 52, n. 9, p. 1–18, 2022.

LIU, H.-C.; XU, D.-H.; DUAN, C.-Y. Pythagorean fuzzy petri nets for knowledge representation and reasoning in large group context. **IEEE Transactions on Systems, Man, and Cybernetics: Systems**, IEEE, v. 51, n. 8, p. 5261–5271, 2021.

LIU, H.-C.; YOU, J.-X.; LI, Z.; TIAN, G. A fuzzy Petri net for knowledge representation and reasoning: a literature review. **Engineering Applications of Artificial Intelligence**, Elsevier, v. 60, p. 45–56, 2017.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, Springer, v. 5, n. 4, p. 115–133, 1943.

MURATA, T. Petri nets: Properties, analysis and applications. **Proceedings of the IEEE**, IEEE, v. 77, n. 4, p. 541–580, 1989.

PAO, Y.-H.; PARK, G.-H.; SOBAJIC, D. J. Learning and generalization characteristics of the random vector functional-link net. **Neurocomputing**, Elsevier, v. 6, n. 2, p. 163–180, 1994.

PARKER, D. Learning-logic (tr-47). **Center for Computational Research in Economics**, Management Science, v. 8, n. TR-47, 1985.

PETERSON, J. L. **Petri Net Theory and the Modelling of systems**. [S.l.: s.n.], 1981.

PETRI, C. A. **Kommunikation mit automaten**. Tese (Doutorado), 1962.

REED, R. Pruning algorithms a survey. **IEEE Transactions on Neural Networks**, IEEE, v. 5, n. 4, p. 740–747, 1993.

ROBLES-ALGARÍN, C.; RESTREPO-LEAL, D.; CASTRO, A. O. Data from multimodal functions based on an array of photovoltaic modules and an approximation with artificial neural networks as a scenario for testing optimization algorithms. **Data in brief**, Elsevier, v. 27, p. 104669, 2019.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. **Psychological review**, American Psychological Association, v. 65, n. 6, p. 386, 1958.

ROWSHANDEL, H.; NICHOLSON, G.; SHEN, J.; DAVIS, C. Characterisation of clustered cracks using an acfm sensor and application of an artificial neural network. **NDT & E International**, Elsevier, v. 98, p. 80–88, 2018.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. **Nature**, Nature, v. 323, n. 6088, p. 533–536, 1986.

SHEN, V. R. L.; CHANG, Y.-S.; JUANG, T. T.-Y. Supervised and unsupervised learning by using Petri nets. **IEEE Transactions on Systems, Man, and Cybernetics**, IEEE, v. 40, n. 2, p. 363–375, 2010.

SI, B.; WANG, J.; YAO, X.; SHI, X.; JIN, X.; ZHOU, X. Multi-objective optimization design of a complex building based on an artificial neural network and performance evaluation of algorithms. **Advanced Engineering Informatics**, Elsevier, v. 40, p. 93–109, 2019.

ULLMAN, J. D. **Elements of ML programming (ML97 ed.)**. [S.l.]: Prentice-Hall, Inc., 1998.

WERBOS, P.; JOHN, P. **Beyond regression : new tools for prediction and analysis in the behavioral sciences.** Tese (Doutorado), 1974.

XU, X.-G.; SHI, H.; XU, D.-H.; LIU, H.-C. Picture fuzzy petri nets for knowledge representation and acquisition in considering conflicting opinions. **Applied Sciences**, Elsevier, v. 9, n. 5, p. 983, 2019.

APÊNDICE A – EXEMPLO DE APÊNDICE

Um apêndice é um documento elaborado pelo autor, diferentemente do anexo. Geralmente, se coloca como apêndice, questionários, códigos de programação, tabelas que tomariam muito espaço no meio do trabalho. Artigos, resumos ou qualquer publicação relacionada ao trabalho podem ser utilizados como apêndice.

O objetivo deste apêndice é mostrar os conjuntos de cores e funções criadas no HTCPN-MLP, para melhorar o entendimento do mesmo e o fato delas seram gerais, podem ser utilizadas para outros objetivos.

Vamos iniciar falando da linguagem, sua característica mais importante é o conceito de ficha colorida.

As primeiras redes de Petri chamadas rede de Petri lugar-transição (RP), só existe um tipo de ficha e a marcação de estado é realizada pela quantidade de fichas em cada lugar da rede. As RPs evoluiram para as redes de Petri coloridas (RPC) e a marcação de estado passou a ser realiada não só pela quantidade de fichas, mas também pelo seu tipo, por exemplo, fichas vermelhas, fichas rosas, fichas azuis etc.

A grande evolução foi quando o conceito de ficha colorida foi generalizado, passando a representar tipos de variáveis, adicionado a isso, uma linguagem de programação CPN ML, a RPC ganhou a capacidade de modelar sistemas complexos e fornecer modelos com um alto nível de abstração e de representação gráfica.

A.1 Conjunto de Cores

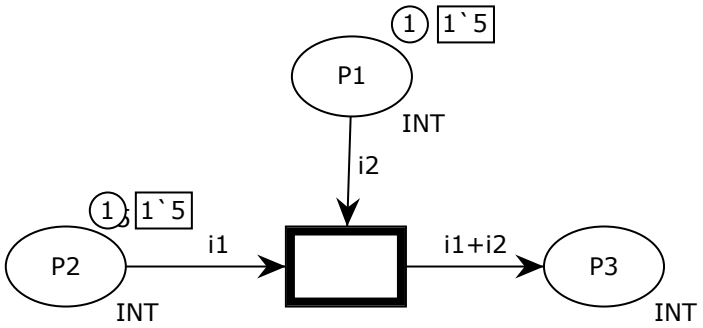
Existe os seguintes conjuntos de cores pré-definidos:

1. colset UNIT é o mais simples, só tem uma cor, representado por ().
2. colset BOOL tem apenas duas cores: verdadeiro e falso.
3. colset INT é o conjunto dos números inteiros.

   No exemplo da Fig.34, temos três lugares que recebem fichas de números inteiros, i1 e i2 são variáveis do tipo INT, ao disparar a transição é retirado as fichas com valores 5 e 6, somado e colocado uma ficha com valor 11 no último lugar.
4. colset INTINF é um conjunto de cores de números inteiros grandes.
5. colset REAL são fichas do conjunto real.

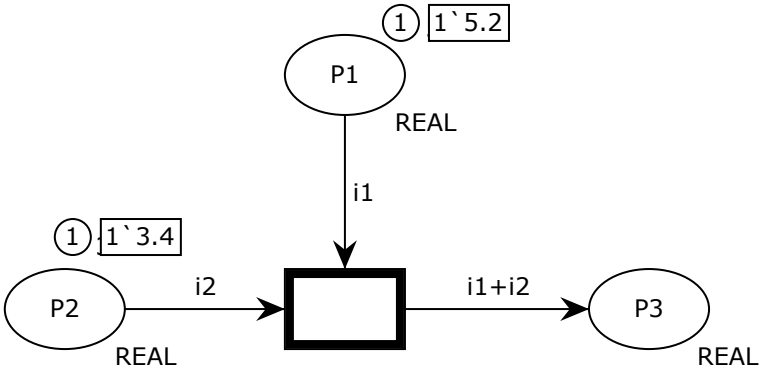
   No exemplo da Fig.35 a rede de Petri realiza a mesma operação (soma de dois números), mas neste exemplo, os números são reais.

Figura 34 – Exemplo do colset INT



Fonte: Elaborada pelos autores.

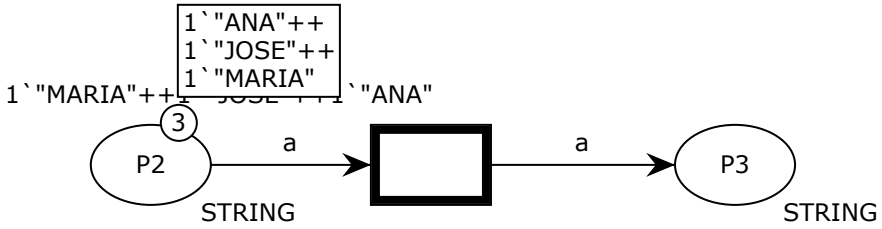
Figura 35 – Exemplo do colset REAL



Fonte: Elaborada pelos autores.

6. colset TIME são adicionadas as fichas uma marcação de tempo.
7. colset STRING são fichas do tipo alfanumérico.

Figura 36 – Exemplo do colset STRING



Fonte: Elaborada pelos autores.

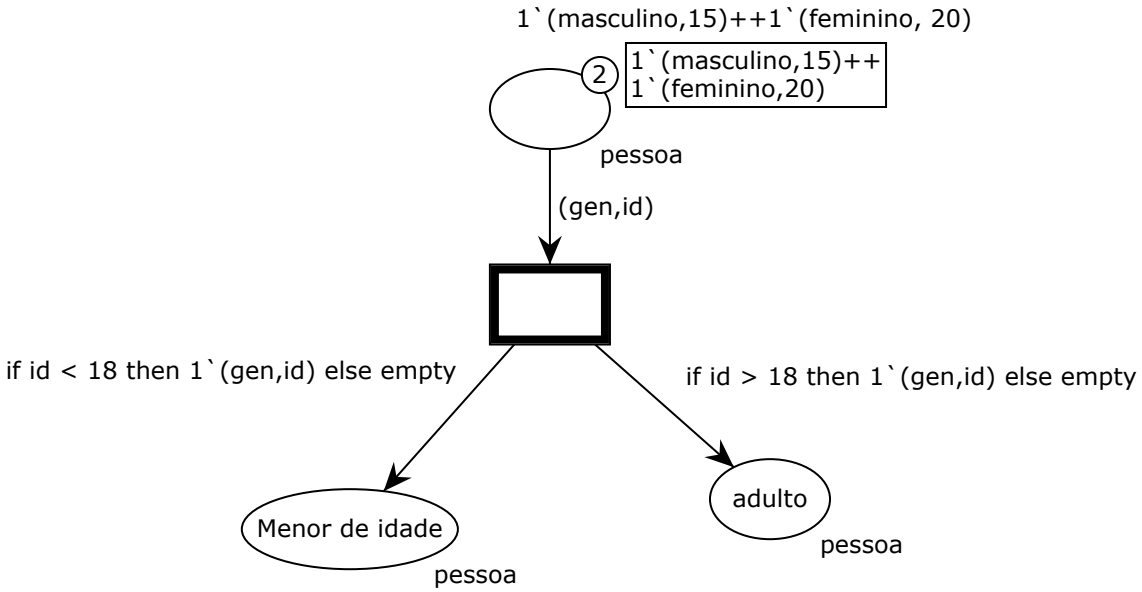
No exemplo da Fig.36 mostra a definição de fichas do tipo alfanumérico.

Além desses conjuntos de cores que já vem definido no CPN tools, o usuário pode criar um conjunto de cores de acordo com a sua necessidade, usando os construtores de tipos: product, record e list.

A.1.1 Construtor product

Com esse construtor podemos criar um produto cartesiano de dois ou mais conjuntos coloridos, ou seja podemos colocar em uma só ficha dois ou mais tipos de conjuntos de cores.

Figura 37 – Exemplo de conjunto de cores usando o construtor product



Fonte: Elaborada pelos autores.

No exemplo da Fig.37 foi criado um conjunto de cores composta por dois tipos: INT e STRING.

colset idade = int with 0 .. 120; (define a idade de uma pessoa);

colset genero = with masculino | feminino; (define o gênero de uma pessoa);

colset pessoa = product genero*idade; (define o conjunto composto que tem duas informações: gênero e idade de uma pessoa);

var id:idade; (variável do tipo idade); var gen: genero; (variável do tipo gênero).

A.1.2 Construtor list

Com esse construtor podemos criar uma lista de conjunto de cores.

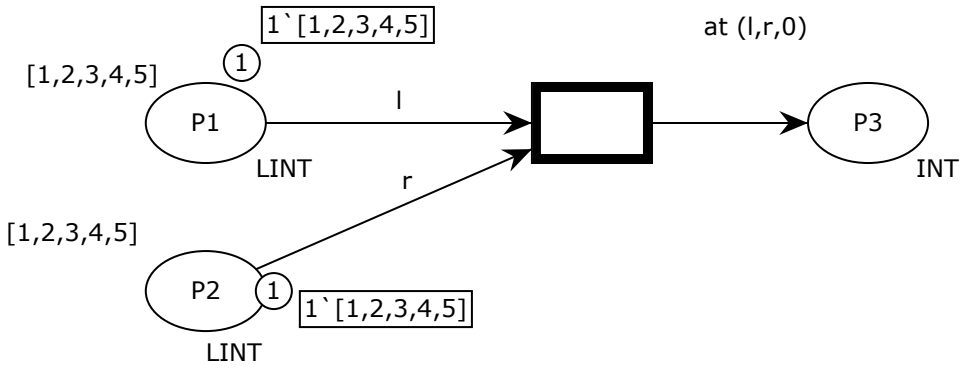
No exemplo da Fig.38 foi criado uma lista de números inteiros.

colset LINT = list INT; (define uma lista de números inteiros);

var l,r: LINT; (variáveis do tipo lista de inteiros);

fun at ([] , r, resultado) = resultado | at (l,r, resultado) = at (tl l, tl r, hd l $*$ hd r + resultado);

Figura 38 – Exemplo de conjunto de cores usando o construtor list



Fonte: Elaborada pelos autores.

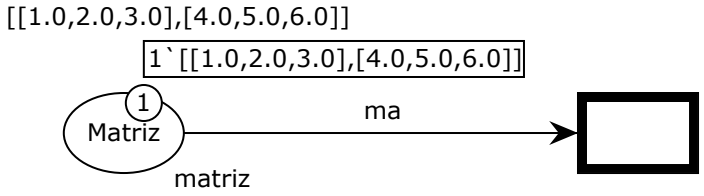
(função at retorna o somatório do produto de cada item da lista).

Uma forma de se criar uma **matriz** no CPN Tools é criando uma lista de uma lista, veja o exemplo a seguir e a figura 39:

colset linha = list RE;

colset matriz = list linha;

Figura 39 – matriz no CPN Tools.



Fonte: Elaborada pelos autores.

Na figura 39 criamos a matriz:

$$A = \begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 4.0 & 5.0 & 6.0 \end{bmatrix}$$

A.2 Estrutura de controle

O CPN tools tem algumas estruturas de controle: escolha entre opções, while do e if the else. No exemplo da Fig. 37 mostra o uso da estrutura if the else, se a idade da pessoa for menor que 18 anos, coloca-se a ficha com seus dados no lugar *Menor de idade* se for maior que 18 anos, coloca-se a ficha com seus dados no lugar *adulto*.

A.3 Funções

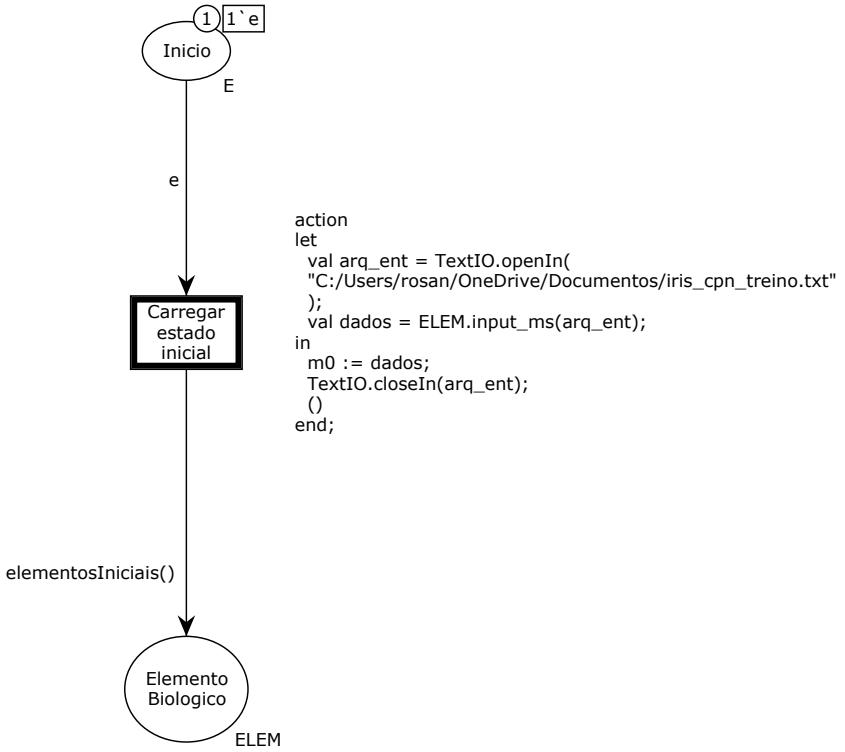
Existe algumas funções definidas no CPN tools:

a) Real.min(r1,r2): minimum of r1 and r2
b) Real.max(r1,r2): maximum of r1 and r2
c) Math.sqrt r: square root of r
d) Math.ln r: natural logarithm
e) Math.exp r: exponential
f) Math.sin r: sine
g) Math.cos r: cosine
h) Math.tan r: tangent
i) Math.arctan r: arc tangent

Além destas funções, o usuário pode criar suas funções, segue abaixo exemplo de funções que eu criei para o HTCPN-MLP.

a) Função carregar banco de dados (fun elementosIniciais)

Figura 40 – Função marcação inicial



Fonte: Elaborada pelos autores.

Esta função carrega um banco de dados no formato txt e transforma-o em fichas no lugar *elemento Biologico* (ver Fig.40).

colset RE = real;

colset ELEM = product RE ∗ RE ∗ RE ∗ RE ∗ RE ∗ RE ∗ RE declare output_col,input_ms;

globref m0 = empty : ELEM ms;

fun elementosIniciais() = (!m0);

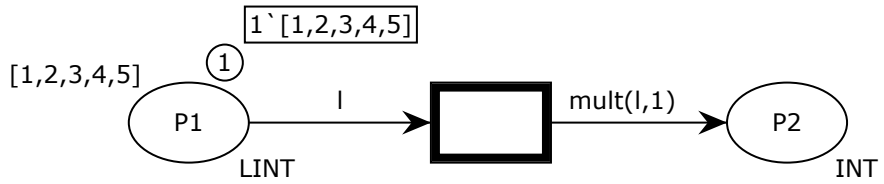
Em uma transição, podemos ter cinco inscrições associadas:

i. Inscrição de nome de transição
ii. Inscrição de guarda
iii. Inscrição de tempo
iv. Inscrição de segmento de código
v. Inscrição de prioridade

Na transição *Carregar estado inicial* tem um segmento de código, o trecho "C:/Users/rosan/OneDrive/Documentos/iris_cpn_treino.txt"é o caminho onde está o arquivo que deve ser carregado (observação: as barras são invertidas).

b) Função multiplicação dos termos de uma lista (fun mult)

Figura 41 – Função multiplicação dos termos de uma lista



Fonte: Elaborada pelos autores.

Esta função multiplica todos os termos de uma lista de números inteiros.

colset LINT = list INT;

var l:LINT;

fun mult ([], resultado) = resultado | mult (l,resultado) = mult (tl l, hd l ∗ resultado)

De maneira análoga, a função *fun soma* soma todos os termos de uma lista de números inteiros. O comando hd retira o primeiro termo de uma lista e tl é o restante da lista.

fun soma([],resultado) = resultado | soma(l,resultado) = soma(tl l,hd l + resultado)

c) Função tangente hiperbólica (fun tgh)

Esta função calcula a tangente hiperbólica de x.

fun tgh (x) =

let

val y = (1.0 - Math.exp(x)) /(1.0 + Math.exp(x))

in

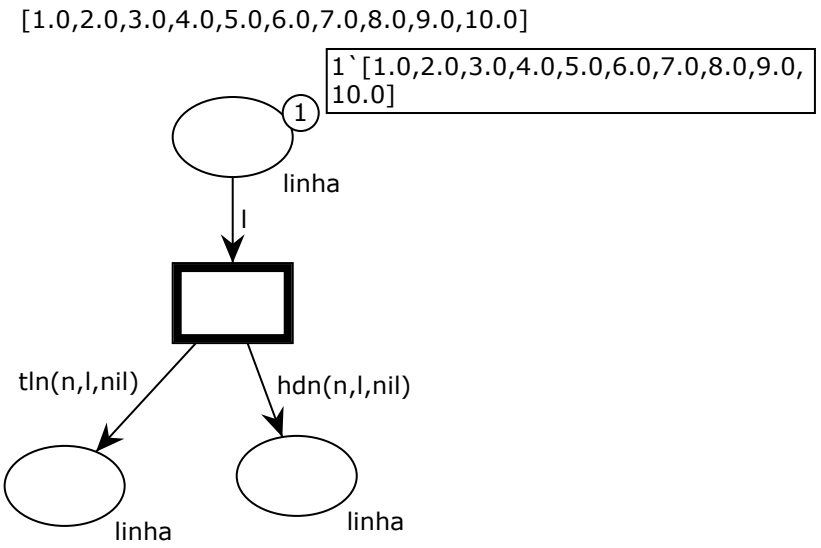
(y)

end

d) Função retira parte de uma lista (fun hdn e tln)

Esta função retira uma parte da lista e a divide em duas e o ususário pode escolher quantos itens irá retirar.

Figura 42 – Função retira parte de uma lista



No exemplo da Fig.42 a função *hdn* retira n números iniciais da lista, n é uma variável que o usuário pode modificar, neste exemplo usei n=6. A função *tln*

e) função retira a primeira e a última linha de uma matriz (fun hd e fun tl) As funções *hd* e *tl* retiram a primeira e a última linha, respectivamente de uma matriz.

f) a função *multm* multiplica o primeiro termo da primeira linha com todos os termos da segunda linha (ver figuras 43 e 44).

colset linha = list RE; colset matriz = list linha;

fun multm ([], [], vetorresposta:linha)=rev(vetorresposta) | multm ([], v, vetorresposta:linha)=rev(vetorresposta) | multm (l, [], vetorresposta:linha)=rev(vetorresposta) | multm (l, v,vetorresposta) = let val el = hd l val ev = hd v

in multm(l, tl v, (ev * el)::vetorresposta) end

g) Duas funções foram criadas *ta* e *tar* para dividir uma lista em duas. Através da

Figura 43 – Função multiplica termo de uma matriz (a)

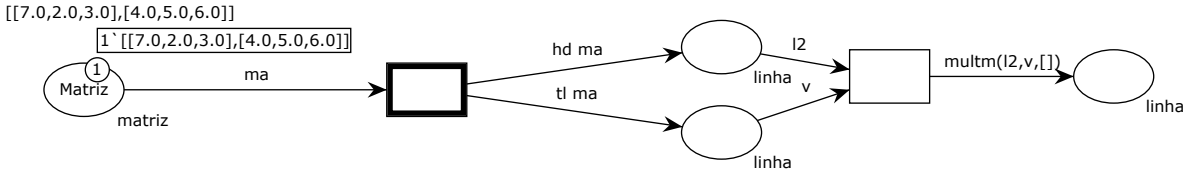
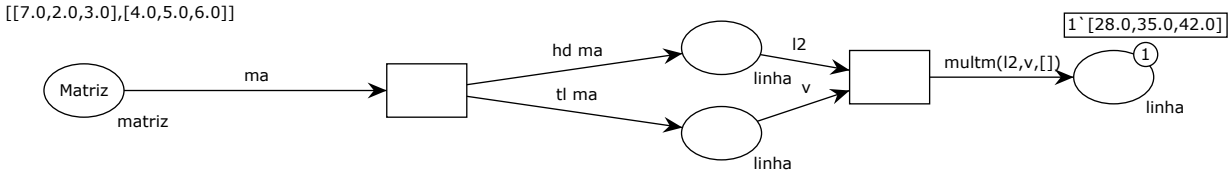


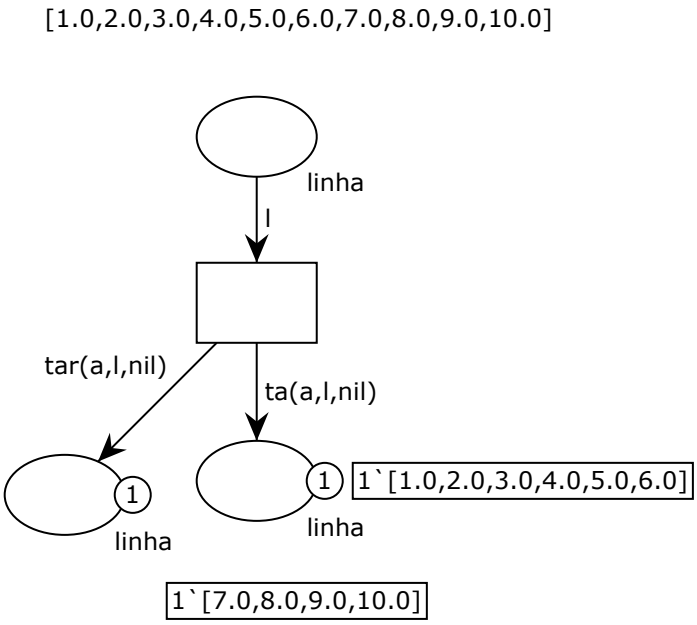
Figura 44 – Função multiplica termo de uma matriz (b)



variável *a* o usuário escolhe em qual termo esta lista será dividida, no exemplo abaixo *a* é igual a 6, ou seja, a primeira lista terá os 6 primeiros termos da lista original e a segunda lista terá do sétimo termo até o último (ver figura).

colset linha = list REAL; val a = 6; fun ta (0, l, resultado:linha) = rev(resultado) | ta (ni,l, resultado) = let val el = hd l in ta (ni-1,tl l, (el::resultado)) end fun tar (0, l, resultado:matriz) = rev(resultado) | tar (ni,l, resultado) = let val el = hd l val et = tl l in tar (ni-1,tl l, if ni=1 then (et::resultado)else empty) end

Figura 45 – Função que divide uma lista em duas



Esses são alguns conjuntos de cores e funções que eu criei no CPN Tools para implementar minha MLP e estou compartilhando elas para você leitor, espero que elas possam ser úteis

ou fonte de inspiração para que você faça suas próprias funções.