

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus Dias Gama

**Uma abordagem baseada na Simulação de
Redes de Petri Coloridas para avaliar o Speedup
de uma Rede Neural Artificial**

Uberlândia, Brasil

2021

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus Dias Gama

**Uma abordagem baseada na Simulação de Redes de
Petri Coloridas para avaliar o Speedup de uma Rede
Neural Artificial**

Trabalho de conclusão de curso apresentado
à Faculdade de Computação da Universidade
Federal de Uberlândia, como parte dos requi-
sitos exigidos para a obtenção título de Ba-
charel em Ciência da Computação.

Orientador: Stéphane Julia

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2021



UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Faculdade de Computação

Av. João Naves de Ávila, nº 2121, Bloco 1A - Bairro Santa Mônica, Uberlândia-MG,
CEP 38400-902

Telefone: (34) 3239-4144 - <http://www.portal.facom.ufu.br/> facom@ufu.br



ATA DE DEFESA - GRADUAÇÃO

Curso de Graduação em:	Ciência da Computação: Bacharelado - Integral				
Defesa de:	Projeto de Graduação 2 - GBC082				
Data da defesa:	04/06/2021	Hora de início:	14:43	Hora de encerramento:	16:45
Matrícula do Discente:	11711BCC012				
Nome do Discente:	MATHEUS DIAS GAMA				
Título do Trabalho:	Uma abordagem baseada na Simulação de Redes de Petri Coloridas para avaliar o Speedup de uma Rede Neural Artificial				

Reuniu-se através da plataforma de Conferência Web disponibilizada pelo Microsoft Teams - utilizada como alternativa à reunião presencial, considerando as recomendações do Ministério da Saúde e a pandemia COVID-19, a Banca Examinadora, designada pelo Colegiado do Curso de Graduação em Ciência da Computação: Bacharelado - Integral, assim composta: Professores: Carlos Roberto Lopes - FACOM/UFU; Márcia Aparecida Fernandes - FACOM/UFU; Stéphane Julia - FACOM/UFU orientador do candidato.

Iniciando os trabalhos, o presidente da mesa, Dr. Stéphane Julia, apresentou a Comissão Examinadora e o candidato, agradeceu a presença do público, e concedeu ao discente a palavra, para a exposição do seu trabalho. A duração da apresentação do discente e o tempo de arguição e resposta foram conforme as normas do curso.

A seguir o(a) senhor(a) presidente concedeu a palavra, pela ordem sucessivamente, aos(às) examinadores(as), que passaram a arguir o(a) candidato(a). Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando o(a) candidato(a):

(X) Aprovado Nota [95]

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.



Documento assinado eletronicamente por **Stéphane Julia, Professor(a) do Magistério Superior**, em 08/06/2021, às 12:25, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do [Decreto nº 8.539, de 8 de outubro de 2015](#).



Documento assinado eletronicamente por **Márcia Aparecida Fernandes, Professor(a) do Magistério Superior**, em 08/06/2021, às 13:59, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do [Decreto nº 8.539, de 8 de outubro de 2015](#).



Documento assinado eletronicamente por **Carlos Roberto Lopes, Professor(a) do Magistério Superior**, em 09/06/2021, às 11:44, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do [Decreto nº 8.539, de 8 de outubro de 2015](#).



A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **2813530** e o código CRC **6CAC32D1**.

Referência: Processo nº 23117.035994/2021-14

SEI nº 2813530

Agradecimentos

Agradeço ao meu orientador Doutor Stéphane Julia pelo auxílio e bons conselhos durante o desenvolvimento do trabalho. Agradeço também, à Professora Doutora Rita Maria da Silva Julia pela assistência e apoio prestados durante a elaboração da produção. Finalmente, agradeço aos meus amigos e familiares, que estiveram ao meu lado durante a pesquisa.

Resumo

Dentre as várias abordagens consagradas para análise de desempenho de algoritmos em termos de tempo de execução, destacam-se a análise assintótica, as técnicas de recorrências e a análise probabilística. Entretanto, há algoritmos que apresentam certas peculiaridades que tornam o uso dessas técnicas puramente matemáticas de avaliação de desempenho inadequadas ou excessivamente árduas. É o caso dos algoritmos distribuídos em que, dependendo da complexidade da política de distribuição utilizada, a avaliação por meio de métodos analíticos do efeito de um gradual incremento de processadores no seu tempo de execução pode tornar-se impraticável. Diante disso, este projeto propõe uma abordagem visual e formal, baseada em simulações automáticas de modelos de Redes de Petri Coloridas Hierárquicas no ambiente gráfico *Colored Petri Nets Tools*, para avaliar o tempo de execução de algoritmos distribuídos usados em Inteligência Artificial. A abordagem proposta será validada por meio de cálculo dos seguintes parâmetros associados aos algoritmos usados como estudo de caso: o tempo de execução e o *speedup*, que é a relação entre o tempo gasto para executar um algoritmo com um único processador e o tempo gasto para executar o mesmo algoritmo com um número N de processadores.

Palavras-chave: Redes de Petri Coloridas, CPN Tools, Redes Neurais Artificiais, Speedup, Simulação.

Lista de ilustrações

Figura 1 – Exemplo de Rede de Petri apresentada em (CARDOSO; VALETTE, 1997)	12
Figura 2 – Exemplo de funcionamento de uma rede de Petri.	14
Figura 3 – Elementos básicos de uma rede de Petri Colorida	15
Figura 4 – Exemplo de funcionamento de uma rede de Petri.	16
Figura 5 – Exemplo da Evolução Dinâmica entre uma Rede e uma Sub-Rede. . . .	17
Figura 6 – Exemplo de Comunicação entre Duas Sub-Redes	19
Figura 7 – Interface do <i>Software</i> CPN Tools	19
Figura 8 – Célula neural artificial proposta em (HRISTEV, 1998)	20
Figura 9 – Perceptron Multicamadas (TOMAZ et al., 2018)	21
Figura 10 – Modelo em RdP para representação de um algoritmo de aprendizado em RN (HANZÁLEK, 2003)	28
Figura 11 – Exemplo de uma rede neural multicamadas (2-4-4-2)	28
Figura 12 – RdPCH construída no CPN Tools	30
Figura 13 – Expansão da transição t0 da Figura 12	30
Figura 14 – Expansão das transições t1 e t2 da Figura 12	31
Figura 15 – Expansão da transição t3 da Figura 12	32
Figura 16 – Expansão da transição t4 da Figura 12	32
Figura 17 – Expansão da transição t5 da Figura 12	33
Figura 18 – Expansão da transição t6 da Figura 12	33
Figura 19 – Expansão das transições t7 e t9 da Figura 12	34
Figura 20 – Expansão das transições l1 a l8 da Figura 19	34
Figura 21 – Expansão da transição t8 da Figura 12	36
Figura 22 – RdPCH que representa o uma PMC realizando processamento em mini-batches	37
Figura 23 – Gráfico de simulação do modelo com processamento on-line	39
Figura 24 – Gráfico que mostra o speedup do modelo utilizando do processamento on-line com N processadores	40
Figura 25 – Gráfico de simulação do modelo com processamento em mini-batches . .	41
Figura 26 – Gráfico que mostra o speedup do modelo utilizando do processamento em <i>Mini-batches</i> com N processadores	42
Figura 27 – Gráfico de simulação do modelo com processamento em batches	43
Figura 28 – Gráfico que mostra o speedup do modelo utilizando do processamento em <i>batches</i> com N processadores	43
Figura 29 – Gráfico apresentando a quantidade de transições disparadas a cada momento da simulação	44

Lista de abreviaturas e siglas

RdP	Redes de Petri
RdPC	Redes de Petri Coloridas
RdPCH	Redes de Petri Coloridas Hierárquicas
RdPEG	Redes de Petri Estocástica Generalizada
RdPE	Redes de Petri Estocástica
RNA	Rede Neural Artificial
PMC	Perceptron de Múltiplas Camadas

Sumário

1	INTRODUÇÃO	9
1.1	Objetivos	10
1.2	Justificativa	10
1.3	Metodologia	11
2	FUNDAMENTAÇÃO TEÓRICA	12
2.1	Redes de Petri	12
2.2	Redes de Petri Coloridas	15
2.3	Redes Neurais Artificiais	20
2.4	Redes Neurais Multicamadas	21
3	TRABALHOS CORRELATOS	23
3.1	Análise de desempenho de softwares baseada em métodos empíricos	23
3.2	Análise de complexidade de Redes Neurais Artificiais utilizando abordagens puramente analíticas	24
3.3	Redes de Petri para a modelagem de problemas	24
3.4	Análise de desempenho de <i>softwares</i> baseada em ferramentas de modelagem visual	25
4	MODELO DE SIMULAÇÃO DE UMA REDE NEURAL ARTIFICIAL	26
4.1	Modelo de Hanzalek	26
4.2	Modelo Hierárquico	29
4.3	Adaptação do Modelo para Processamento em Lotes	35
4.4	Modelo de Alocação de Recursos	37
5	SIMULAÇÃO DOS MODELOS EM REDE DE PETRI DA RNA . .	38
5.1	Simulação do Processamento <i>On-line</i>	38
5.2	Simulação do Processamento da RNA em <i>Mini-batches</i>	40
5.3	Simulação do Processamento em <i>Batches</i>	41
5.4	Análise dos Monitores	44
6	CONSIDERAÇÕES FINAIS	46
	REFERÊNCIAS	48

1 Introdução

São inúmeros os problemas que despertam o interesse da comunidade científica pela relevância teórica e prática cujas soluções ainda não foram encontradas - ou, então, foram sugeridas de maneira pouco eficiente - . No cenário da Computação, para que uma solução proposta por meio de um algoritmo seja eficiente, devem ser considerados dois aspectos: a memória requerida e o tempo de execução gasto ([CORMEN, 2009](#)). Com a evolução e a desoneração do *hardware*, o primeiro aspecto tem apresentado um desafio menor quando comparado ao segundo. Dessa forma, a fim de avaliar o nível de qualidade de soluções baseadas em algoritmos, torna-se imprescindível contar com abordagens apropriadas de avaliação do tempo de execução. Dentre as abordagens analíticas consagradas disponíveis para tal fim, destacam-se ([CORMEN, 2009](#)): a análise assintótica, a qual mede o impacto do aumento da dimensão do dado de entrada no tempo de execução; as técnicas que resolvem as recorrências, usadas para avaliar os algoritmos recursivos (método da substituição, método da árvore de recursão ou método Máster); e a análise probabilística, que usa probabilidades de distribuição dos dados de entrada para estimar o tempo de execução.

Entretanto, há algoritmos que apresentam certas peculiaridades que tornam o uso dessas técnicas puramente matemáticas de avaliação de desempenho inadequadas ou excessivamente árduas. É o caso dos algoritmos distribuídos em que, dependendo da complexidade da política de distribuição utilizada, a avaliação por meio de métodos analíticos do efeito de um gradual incremento de processadores no seu tempo de execução pode tornar-se impraticável (o que ocorre, por exemplo, sempre que o tempo de execução depende de atualizações de valores de parâmetros efetuadas via troca de mensagens ([BARBOSA, 1996](#))).

Outros autores propuseram um método empírico baseado na complexidade computacional para entender a escalabilidade de programas ([GOLDSMITH; AIKEN; WILKERSON, 2007](#)). Para este caso em particular, entende-se por escalabilidade a capacidade de ganho de desempenho do programa em função do incremento de novas cargas de trabalho ([COULOURIS, 2012](#)). Em ([BOROVSKA; LAZAROVA, 2007](#)), foi proposto um método empírico baseado na comparação de desempenho de modelos de programação paralelos para avaliar o *speedup*, a eficiência e a escalabilidade usando um algoritmo de busca paralelo. Contudo, tais alternativas apresentam o seguinte inconveniente: elas requerem a implementação dos algoritmos analisados, o que pode ter um efeito perverso particularmente no caso dos algoritmos distribuídos, uma vez que isso demandaria a aquisição prévia de recursos de *hardware* dispendiosos de multi-processamento antes mesmo de saber se a proposta de distribuição investigada, de fato, vale a pena.

Neste trabalho foi desenvolvida uma abordagem visual e formal para avaliar o desempenho de algoritmos em termos de seus tempos de execução, efetuando, para tanto, simulações de modelos baseados em Redes de Petri Coloridas (RdPC) e Redes de Petri Coloridas Hierárquicas (RdPCH) (JENSEN; KRISTENSEN, 2009) no ambiente gráfico CPN Tools (JENSEN et al., 2018). A abordagem proposta será validada por meio de cálculo dos seguintes parâmetros associados aos algoritmos usados como estudo de caso: o tempo de execução e o *speedup*. Serão escolhidos como estudos de caso algoritmos distribuídos utilizados em Inteligência Artificial. Será estudado particularmente o caso de Redes Neurais Artificiais de múltiplas camadas (HASSOUN et al., 1995) que dependem de 2 ou mais processadores (HANZÁLEK, 2003).

1.1 Objetivos

O objetivo geral deste trabalho foi de propiciar uma abordagem para avaliar o desempenho de algoritmos seriais e distribuídos baseada em simulações automáticas de modelos formais baseados em RdPC, abordagem, esta, que prescindia da implementação dos algoritmos analisados.

Para atingir o objetivo geral, foram traçados dois objetivos específicos que são:

- Modelar usando as RdPC os algoritmos a serem analisados usando o ambiente CPN Tools;
- Calcular o *speedup*, que mede a performance relativa de um sistema em relação a outro, quando os algoritmos dependem de mais de um processador (caso distribuído).

Os algoritmos considerados no estudo de caso serão algoritmos usados em Inteligência Artificial baseados em heurísticas e para os quais, particularmente no caso distribuído, não existem abordagens puramente analíticas de análise da função de complexidade. O estudo de Redes Neurais Artificiais ilustrará a abordagem de modelagem, simulação e análise da proposta apresentada neste projeto.

1.2 Justificativa

A principal contribuição científica deste trabalho é a apresentação de uma abordagem visual e formal, baseada nos resultados apresentados em (JÚNIOR et al., 2018) e (JÚNIOR; JULIA; JULIA, 2015), para a análise de desempenho e de complexidade de algoritmos seriais e distribuídos usados em Inteligência Artificial. Tal abordagem será apta a lidar com algoritmos que envolvem comandos condicionais, iterações, e chamadas/retornos recursivos. A proposta se tornará relevante em situações que envolvem algoritmos cujo

tempo de execução dificilmente será passível de ser calculado por métodos analíticos tradicionais, tal como ocorre nos algoritmos distribuídos usados em Inteligência Artificial que dependem de heurísticas complexas. Além disso, o método apresentado será uma alternativa bastante útil para calcular o *speedup* e a eficiência de algoritmos distribuídos sem que seja necessário implementá-los, usando simulações de modelos de RdP.

Acredita-se, também, que a abordagem proposta representará uma ferramenta importante para a análise de alternativas a fim de melhorar o desempenho de algoritmos distribuídos conhecidos. Mais especificamente, a aplicação das propriedades das RdP aos modelos construídos deverá permitir a detecção de novas possibilidades de re-distribuição de tarefas entre os processadores disponíveis, reduzindo o ócio dos mesmos.

1.3 Metodologia

A pesquisa proposta por este trabalho é exploratória, visto que além das fontes bibliográficas acerca do tema elegido, será proposto um estudo de caso para que os modelos em Redes de Petri possam ser monitorados e validados, sendo usados como a sustentação para a solução do problema. Assim, o levantamento bibliográfico juntamente com o estudo de caso tem o papel de explorar os conceitos e analisar as possíveis soluções. Os modelos de simulação produzidos na ferramenta *CPN Tools* serão utilizados ainda para avaliar o desempenho do método proposto.

Serão executadas as seguintes etapas especificadas a seguir para cumprir os objetivos especificados:

- realizar um estudo minucioso sobre o referencial teórico, relacionados com o uso das Redes de Petri na modelagem de algoritmos seriais e distribuídos;
- definir os principais elementos de modelagem dos blocos de comando usados em algoritmos seriais e distribuídos;
- modelar formalmente, usando RdPCH, um exemplo de Rede Neural Artificial de múltiplas camadas processada por um único processador;
- modelar formalmente, usando RdPCH, um exemplo de Rede Neural Artificial de múltiplas camadas processada por mais de 1 processador;
- realizar de forma experimental a simulação no CPN Tools dos modelos produzidos, considerando: tempo de execução, *speedup* e eficiência;

2 Fundamentação Teórica

A seguir serão descritos conceitos relacionados a Redes de Petri, Redes de Petri Coloridas, Redes Neurais Artificiais e Redes Neurais Multicamadas, com o intuito de propiciar uma fundamentação necessária para um melhor entendimento do trabalho que foi desenvolvido.

2.1 Redes de Petri

Em sua dissertação de doutorado, Carl Adam Petri apresentou um tipo de grafo bipartido com estados associados, com o objetivo de estudar a comunicação entre autômatos ([MURATA, 1989](#)). Após Petri apresentar seus primeiros modelos, alguns pesquisadores, com o intuito de homenagear sua descoberta, os denominaram de Redes de Petri (RdP). As RdP são ferramentas matemáticas, gráficas e formais que permitem a modelagem, a análise, o controle e a supervisão de sistemas dinâmicos, produzindo uma abordagem discreta dos mesmos.

Os elementos básicos das RdP são:

- o lugar, representado por um círculo, que pode modelar um estado parcial, uma condição, uma espera, ou um procedimento;
- a transição, representada por uma barra ou retângulo, que modela um evento de início ou fim de um estado;
- o arco, representado por um arco direcionado, que liga transição(ações) a(aos) lugar(res) e vice-versa;
- a ficha ou marca, representada por um ponto em um lugar, que modela a ocorrência de um estado.

Vale ressaltar que um lugar nunca é ligado a outro diretamente e uma transição nunca é ligada a outra diretamente, ou seja, um lugar sempre está ligado a uma ou mais

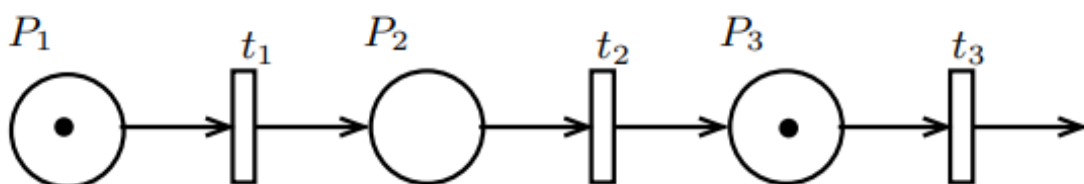


Figura 1 – Exemplo de Rede de Petri apresentada em ([CARDOSO; VALETTE, 1997](#))

transições por meio de arcos e uma transição, por sua vez, sempre está ligada a um ou mais lugares também por intermédio dos arcos (MURATA, 1989).

Além disso, os arcos podem possuir inscrições, representando a quantidade de fichas necessárias para a sensibilização de uma transição no caso em que o arco liga um lugar a uma transição, ou a quantidade de fichas que serão geradas em um lugar após o disparo de uma transição quando a inscrição estiver em um arco que liga uma transição a um lugar.

As RdP são ferramentas de modelagem dinâmica, pois permitem acompanhar a evolução do sistema modelado por meio da noção de fluxos de fichas. A mudança da ficha de um lugar ao outro depende do modelo de RdP utilizado que definirá uma política de disparo específica. Por exemplo, se o modelo é uma RdP temporizada com tempos de execução de operações associados com as transições do modelo, os disparos de transições não são instantâneos e dependem das durações associadas às transições do modelos. Porém, se o modelo é uma RdP Predicado/Transição, os disparos de transições são instantâneos, mas ocorrem somente quando são satisfeitas condições associadas às transições do modelo e que correspondem a Predicados de uma lógica de primeira ordem.

A Figura 2 ilustra o funcionamento da rede de Petri. No início, o lugar $P2$ possui duas fichas e o lugar $P1$ possui uma ficha. Dessa forma, as transições A e B estão habilitadas, como mostra a figura 2a. Considere que a transição B será disparada, produzindo assim uma ficha no lugar $P2$ (figura 2b). Em seguida, a transição C estará apta para ser disparada (figura 2c). Ao ser disparada, a transição C consome três fichas de seu lugar de entrada e produz uma ficha em seu lugar de saída, como ilustra a figura 2d.

A marcação de uma rede de Petri, denotada por M , diz respeito a distribuição de fichas nos lugares. Durante a execução da rede, o número de fichas pode mudar. De acordo com o conjunto de marcações acessíveis a partir da marcação inicial (M_0), são definidas algumas propriedades comportamentais (reagrupadas sob o nome genérico de boas propriedades (CARDOSO; VALETTE, 1997)). Tais propriedades são definidas por (MURATA, 1989) e apresentadas a seguir:

- **Alcançabilidade:** essa propriedade indica a possibilidade de alcançar uma determinada marcação. Uma marcação M_n é dita alcançável a partir da marcação M_0 se existe uma sequência finita de disparo de transições que conduza de M_0 para M_n . Essa propriedade garante que certos estados serão alcançados ou não;
- **Limitabilidade:** uma rede de Petri é dita limitada se o número de fichas em cada lugar da rede não exceder um inteiro positivo k para qualquer marcação alcançável a partir de M_0 . Neste caso, a rede é dita k -limitada. Se a rede for limitada ao inteiro 1, então diz-se que ela é binária;

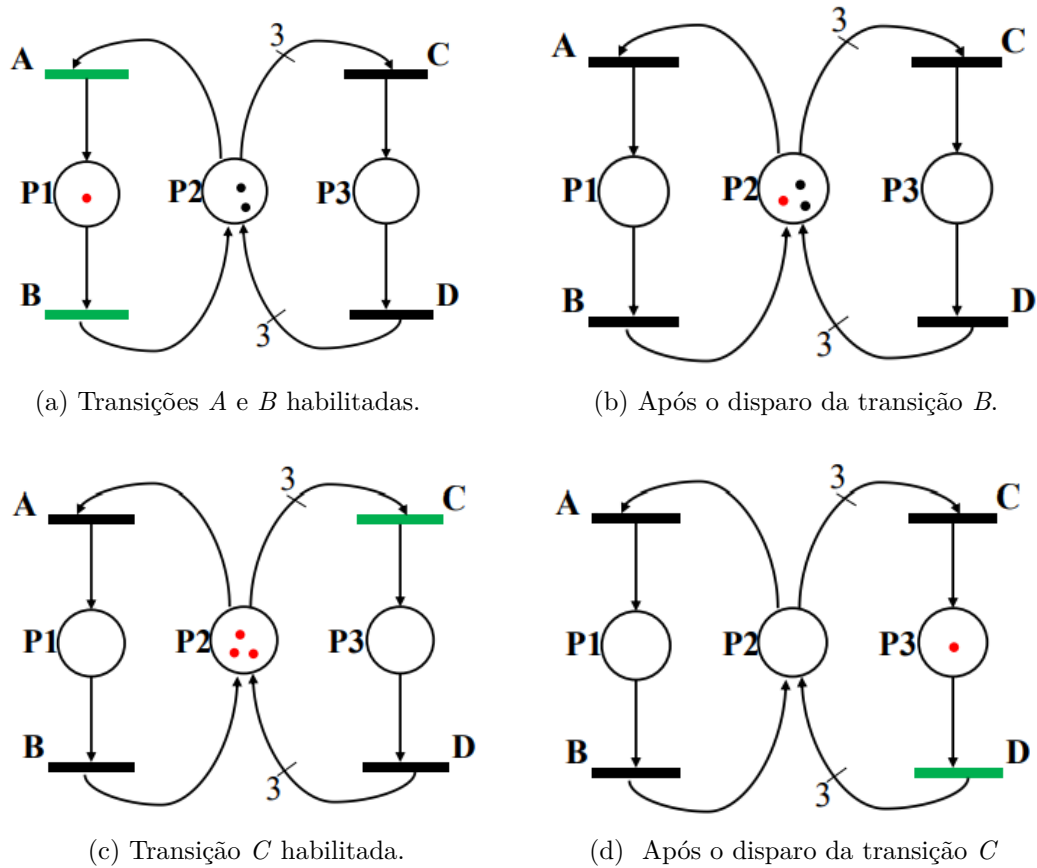


Figura 2 – Exemplo de funcionamento de uma rede de Petri.

- **Vivacidade:** uma rede de Petri é dita viva se todas as suas transições são vivas. Uma transição t é dita viva se para cada marcação alcançável da rede, existe uma sequência de disparo S que sensibiliza t . Dessa forma, uma rede de Petri viva é uma rede onde todas as suas transições são disparáveis. Essa propriedade garante que o sistema é livre de *deadlock*;
- **Reiniciabilidade:** uma rede de Petri é dita reiniciável se, para qualquer marcação M , M_0 é alcançável a partir de M . Em outras palavras, a rede é reiniciável se é sempre possível voltar para a marcação inicial através de uma sequência de disparos, seja qual for a marcação considerada.

A potencialidade da modelagem de sistemas e algoritmos por meio das RdP destaca-se em relação a outras técnicas mais tradicionais, como os autômatos, pelo fato delas permitirem modelar restrições de alocação de recursos, de sincronização e de paralelismo, dentre outras. Sabe-se que, por meio dos autômatos, não é possível modelar de forma explícita restrições de paralelismo e nem de sincronização. Em virtude das RdP permitirem a modelagem de diversas restrições, elas podem gerar modelos mais compactos que os autômatos, permitem abordagens visuais e formais, e representam a dinâmica de sistema com características de paralelismo, concorrência e sincronização.

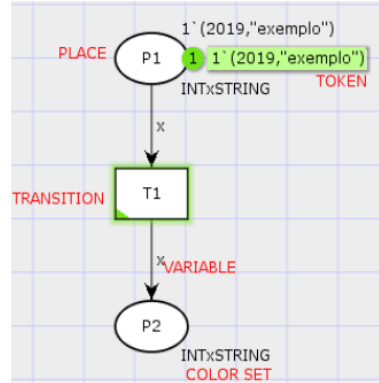


Figura 3 – Elementos básicos de uma rede de Petri Colorida

2.2 Redes de Petri Coloridas

Segundo [Jensen e Kristensen \(2009\)](#), uma Rede de Petri Colorida (RdPC) é uma rede na qual cada lugar possui um tipo abstrato de dado que define um conjunto U . Tal conjunto é um universo contendo todos os possíveis valores pertencentes a este tipo. Um multi-conjunto sobre U é então um mapeamento $f : U \rightarrow \mathbb{N}$ que atribui para cada elemento $u \in U$ um número natural $f(u)$. Cada elemento $u \in U$ especifica um valor do tipo do lugar representado por U . Além disso, um arco em uma RdPC pode ter uma etiqueta (ou rótulo) associada a ele. Esta etiqueta é uma expressão com algumas variáveis que são avaliadas sobre um multi-conjunto. Uma transição pode ter uma Guarda ou Expressão de guarda associada a ela. Esta Guarda é uma expressão booleana que pode ter variáveis que correspondem às etiquetas dos arcos de entrada e saída da referida transição. Uma vez que cada lugar de uma RdPC possui um tipo (cor) associado a ele, as fichas que são inseridas nesse lugar devem estar em conformidade com seu tipo que deve ser previamente declarado. Com isso, as RdPC são uma linguagem de modelagem gráfica que permite a associação da potencialidade das RdP - representação da sincronização e da concorrência em sistemas distribuídos, por exemplo - com o formalismo das linguagens de programação funcionais.

A RdPC da Figura 3 possui dois lugares denominados $P1$ e $P2$. Os lugares possuem o tipo de dado (*color set*) $INTxSTRING$, assim como a variável x do mesmo tipo associada aos arcos da rede. Este tipo de dado é formado pelo produto cartesiano de dois tipos: INT (inteiros) e $STRING$ (cadeia de caracteres). Os lugares da rede aceitam apenas fichas do mesmo tipo, ou seja, fichas do tipo $INTxSTRING$. Nesse sentido, a inscrição $1'(2019, "exemplo")$ corresponde a uma ficha cujos atributos são dados pelo valor inteiro (2019) e pela *string* ("exemplo").

No exemplo da figura 4, a transição $T1$ será habilitada apenas se em seu lugar de entrada ($P1$) existir pelo menos uma ficha. Durante o disparo de uma transição no modelo, as variáveis dos arcos de entrada serão substituídas pelo valor da ficha. Para

exemplificar, considere o exemplo da figura 4a que ilustra o disparo da transição $T1$. No momento do disparo o valor (2019,"exemplo") é associado com a variável x . Após disparar, $T1$ produzirá uma ficha em $P2$, como ilustra a figura 4b.

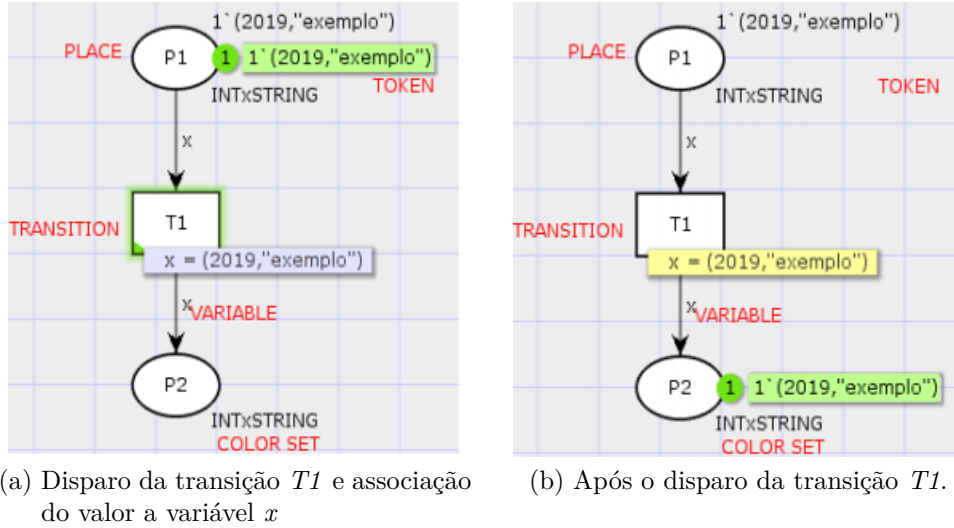
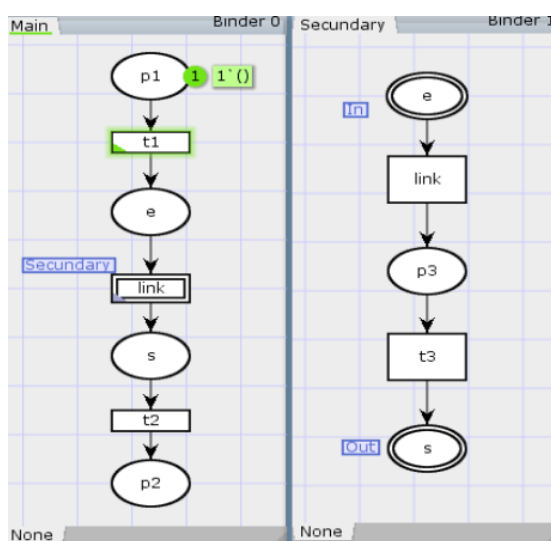


Figura 4 – Exemplo de funcionamento de uma rede de Petri.

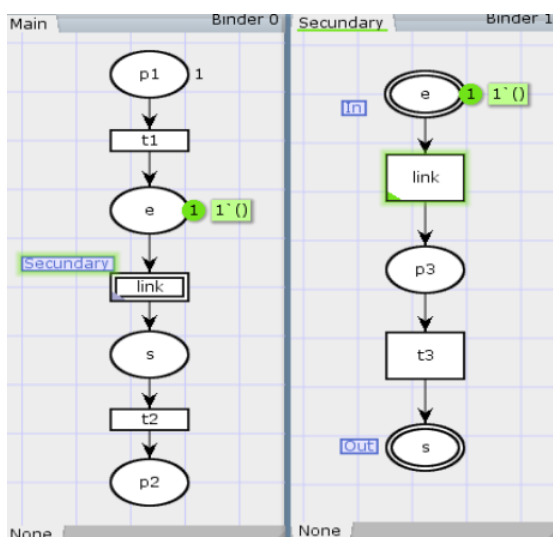
Um ambiente de modelagem e simulação que tem contribuído muito com a “popularização” do uso das RdPC e de suas extensões é o CPN Tools (JENSEN et al., 2018). Esse ambiente é composto por um conjunto de ferramentas que permitem ações, tais como editar, simular, analisar espaços de estados e avaliar o desempenho dos modelos criados. Na ferramenta, o usuário pode trabalhar explorando um ambiente gráfico com diversos recursos. Além disso, é possível o uso combinado da programação funcional com o modelo a ser criado de forma bem intuitiva e prática. Outro diferencial dessa ferramenta é que ela é distribuída gratuitamente.

Uma característica interessante da implementação das redes de Petri Coloridas por meio da ferramenta CPN Tools é a possibilidade de estruturar os modelos em diferentes módulos. O conceito de módulos em RdPC é baseado em um mecanismo de estruturação hierárquica. A ideia básica da hierarquia por trás das RdPC é permitir a construção de um amplo modelo combinando um número de pequenas redes de Petri Coloridas em um único modelo (JENSEN; KRISTENSEN, 2009). De acordo com (AALST; STAHL, 2011), essa característica facilita a modelagem de sistemas grandes e complexos, tais como sistemas de informação e de processos de negócio.

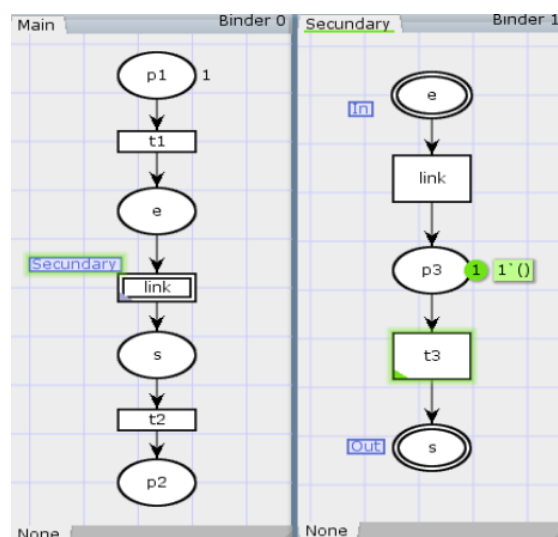
Na figura 5, é possível observar a evolução dinâmica entre a rede principal e a sub-rede utilizando o mecanismo de estruturação hierárquico do CPN Tools. Considerando o modelo da Figura 5a, no qual a marcação inicial mostra uma ficha no lugar $p1$ sem cor pré-definida, observa-se que há uma rede principal chamada *Main* e uma subrede chamada *Secondary*. A transição abstrata *link* estabelece o elo entre elas. Nota-se que, na sub-rede



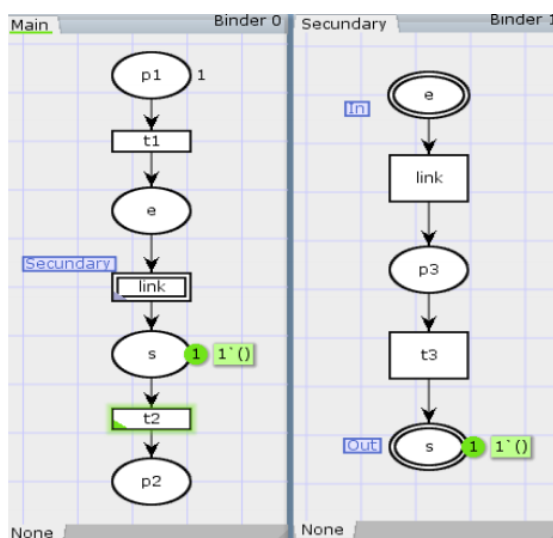
(a)



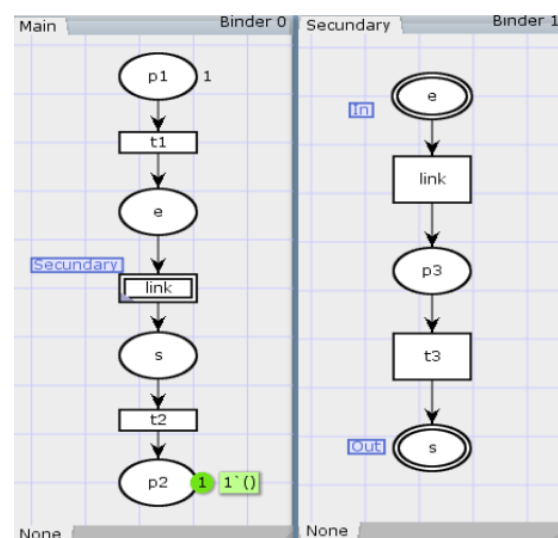
(b)



(c)



(d)



(e)

Figura 5 – Exemplo da Evolução Dinâmica entre uma Rede e uma Sub-Rede.

Secondary há um lugar com borda dupla (*e*) associado a um *socket* de entrada (*In*) e há outro lugar com borda dupla (*s*) associado a um *socket* de saída (*Out*). Conforme mostrado na Figura 5b, quando uma ficha é produzida no lugar *e* da rede *Main*, ela também é produzida no lugar *e* da sub-rede *Secondary*, pela função da transição abstrata *link*. Em seguida, com o disparo da transição *link* da sub-rede *Secondary*, é produzida uma ficha no lugar *p3* (Figura 5c). Ao disparar a transição *t3*, a ficha que é produzida no lugar *s* da sub-rede *Secondary*, também é produzida no lugar *s* da rede *Main*, refletindo o elo formado entre a rede principal e a sub-rede (Figura 5d). Finalmente, a transição *t2* da rede *Main* é disparada e a ficha é produzida no lugar *p2* (Figura 5e). Vale destacar que, com a criação da transição abstrata *link*, os *sockets In* e *Out* foram inseridos na sub-rede *Secondary* para permitir a entrada e saída de fichas dessa rede..

A hierarquia das redes de Petri Coloridas oferece um conceito conhecido como *fusion places* (lugares de fusão). Esse conceito permite especificar um conjunto de lugares que são considerados idênticos (KRISTENSEN; CHRISTENSEN; JENSEN, 1998). Isto significa que todos esses lugares representam um único lugar conceitual, ainda que sejam desenhados como vários lugares individuais. Esses lugares são chamados, individualmente, *fusion places*, e um conjunto de *fusion places* é chamado de *fusion set* (conjunto de fusão). Qualquer ação que acontecer a um lugar do conjunto de fusão, também acontece aos outros lugares do mesmo conjunto. Assim, se uma ficha for adicionada/consumida de um dos lugares, uma ficha idêntica também será adicionada/consumida em todos os outros lugares do conjunto de fusão.

A Figura 6a mostra um exemplo no qual duas sub-redes (*Secondary1* e *Secondary2*) se comunicam por meio de lugares de fusão. Quando uma ficha é produzida no lugar de fusão *p3* (*Fusion1*) da sub-rede *Secondary1* (Figura 6b), uma cópia dela também é produzida no lugar de fusão *p3* (*Fusion1*) da sub-rede *Secondary2*. Isso reflete o fato de que o fluxo da ficha passou da sub-rede *Secondary1* para a sub-rede *Secondary2*. É importante observar que tal mudança fez com que a transição *t3* da sub-rede *Secondary2* se tornasse apta ao disparo (Figura 6c). Quando a transição *t3* for disparada, a ficha será produzida no lugar de fusão *p4* (*Fusion2*) de ambas as sub-redes (*Secondary1* e *Secondary2*), indicando que o fluxo da ficha voltará para a sub-rede *Secondary1*.

O *software* possui interface gráfica eficiente, além de trazer à disposição dos usuários todas as funcionalidades necessárias para a edição de uma RdPC, desde paletas de criação, paletas de análise e simulação, e até paletas de configurações (estilo/preferências) para o modelo. A Figura 7 ilustra a interface do *Software CPN Tools* bem como todas as opções de funcionalidades disponíveis para uso no menu da ferramenta.

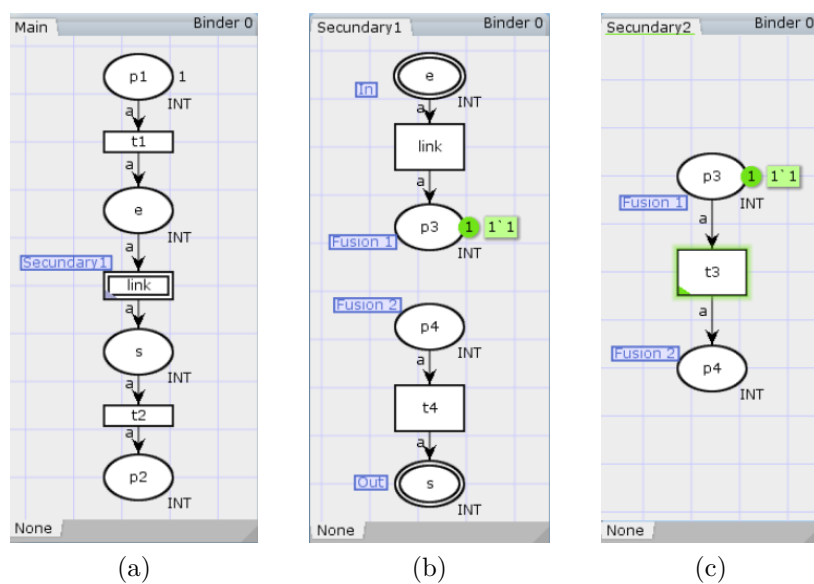


Figura 6 – Exemplo de Comunicação entre Duas Sub-Redes

Figura 7 – Interface do *Software* CPN Tools

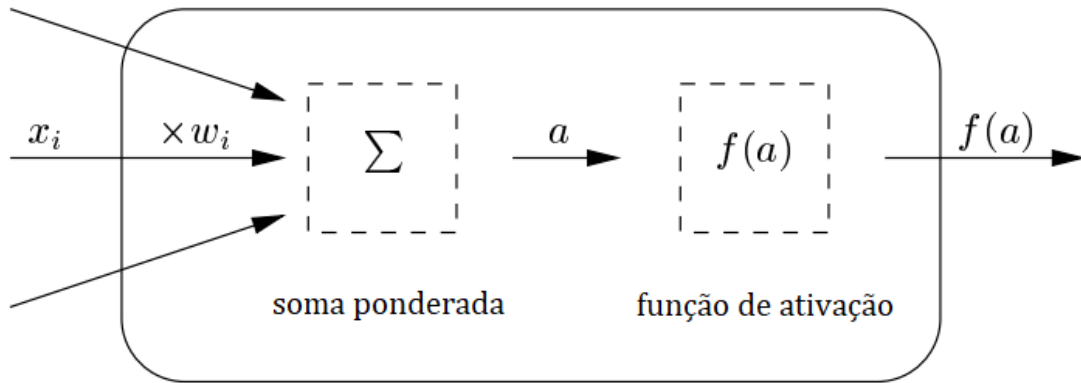


Figura 8 – Célula neural artificial proposta em (HRISTEV, 1998)

2.3 Redes Neurais Artificiais

Redes Neurais Artificiais são modelos computacionais de Inteligência Artificial que se inspiram no sistema nervoso dos seres humanos (AMARI et al., 2003). Esses modelos são formados por um conjunto de elementos individuais denominados neurônios que se conectam possibilitando o armazenamento e transmissão de dados de entrada.

O modelo proposto do neurônio proposto em (HRISTEV, 1998), ilustrado na Figura 8, é composto pelos seguintes elementos:

- Sinais de entrada $\{x_0, x_1, \dots, x_n\}$: sinais que representam os dados que serão processados;
- Pesos sinápticos $\{w_0, w_1, \dots, w_n\}$: os valores utilizados para ponderar cada entrada do neurônio, quantificando a relevância de cada entrada;
- Combinador linear Σ : Realiza a operação de soma dos valores de entrada ponderados pelos seus respectivos pesos sinápticos;
- Função de ativação f : Limita o sinal de saída do neurônio a valores adequados ao problema tratado pelo modelo neural;
- Sinal de saída $f(a)$: Valor final produzido pelo neurônio em relação a seu conjunto de entradas.

Dessa forma, o conhecimento adquirido e mantido por uma RNA é inerente às conexões entre os diversos neurônios que a compõe (HAYKIN, 2007). Assim, as RNAs possuem uma alta capacidade de aprender e modelar problemas não-lineares e complexos, o que as permitem serem úteis nas tarefas de reconhecimento/classificação de padrões, no agrupamento de dados, em sistemas de previsão, na otimização de sistemas, entre outras.

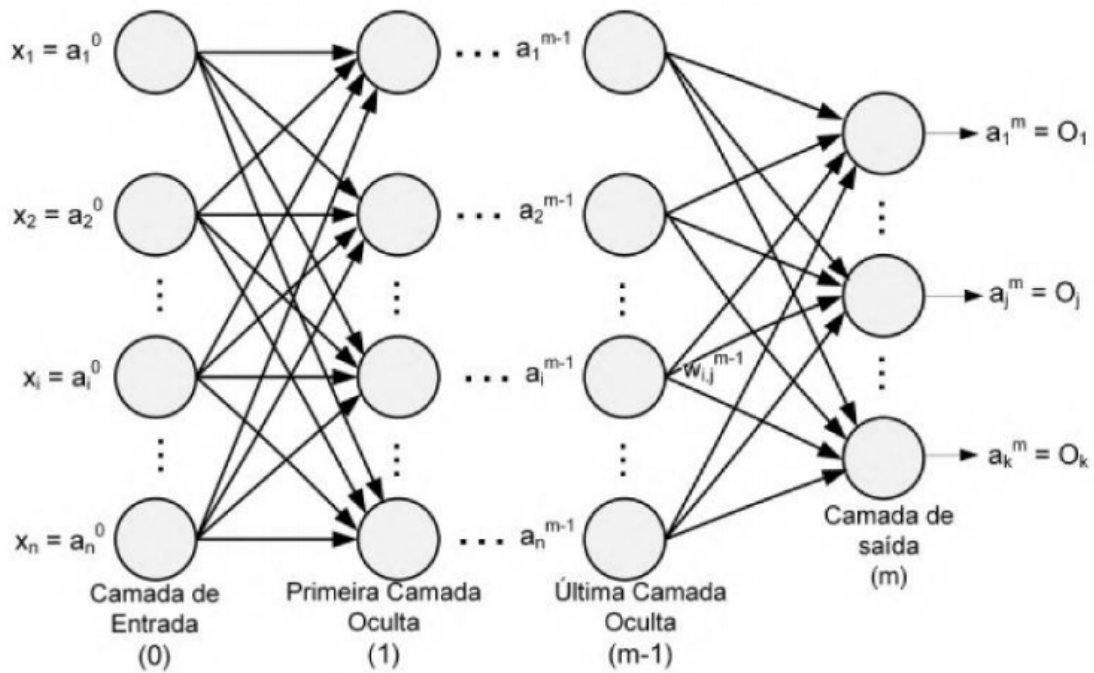


Figura 9 – Perceptron Multicamadas (TOMAZ et al., 2018)

2.4 Redes Neurais Multicamadas

As redes neurais artificiais são aplicadas com grande nível de sucesso em vários tipos de problemas. Ainda que existam diversas arquiteturas de redes neurais, a arquitetura multicamadas é a mais utilizada na literatura. Entre os motivos que contribuem para sua popularidade, estão a sua capacidade de aproximação universal e sua flexibilidade para abstrair de um único algoritmo de aprendizado, diversas soluções satisfatórias para uma vasta classe de problemas. São amplamente utilizadas para reconhecimento de padrões (BISHOP et al., 1995), mas também são utilizadas para resolver diversos outros problemas. Em (FERNEDA, 2006), são aplicadas em sistemas de recuperação de informações de grandes bases documentais. Em (FERREIRA et al., 2011), as redes neurais multicamadas são utilizadas como estratégia de previsão de preços de commodities no contexto do agronegócio.

As Perceptron de Múltiplas Camadas (PMCs) são RNAs com arquitetura *feed-forward* (isto é, os sinais de entrada se propagam em um único sentido: da camada de entrada para a camada de saída) que contém mais de uma camada de neurônios processadores (SILVA; SPATTI; FLAUZINO, 2010). Neste caso, as camadas processadoras dispostas entre a camada de entrada e a camada de saída são denominadas camadas ocultas, conforme ilustrado na Figura 9.

O treinamento de um PMC, ou seja, o processo de ajuste de pesos para minimizar o erro, ocorre da seguinte maneira: as respostas produzidas pelas saídas da rede são comparadas com as respectivas respostas desejadas que estejam disponíveis. Mais especi-

ficamente, os desvios (erros) entre as respostas desejadas produzidos pelos m neurônios de saída, durante cada época, nortearão o processo de ajuste de pesos sinápticos. Neste processo, o reajuste ocorrerá sob a estratégia de *backpropagation* (RUMELHART; HINTON; WILLIAMS, 1986), ou seja, os erros produzidos nos neurônios de saída para cada amostra desencadearão o seguinte fluxo: inicialmente, eles serão utilizados para reajustar os pesos das conexões entre a última camada oculta e a camada de saída; na sequência, serão ajustados os pesos das conexões entre a penúltima e a última camada oculta. Os ajustes continuam nessa dinâmica de retropropagação até que se tenham reajustado os pesos entre a camada de entrada e a primeira camada oculta.

Um dos algoritmos de otimização utilizado para encontrar pesos e coeficientes em algoritmos de aprendizado de máquina, como as redes neurais, é o gradiente descendente. O algoritmo utiliza de predições do modelo sobre dados de treinamento para encontrar um erro, que é considerado ao atualizar os pesos do modelo com o intuito de reduzir o erro, fazendo isso ao caminhar pelo gradiente da função de cálculo do erro, procurando um valor mínimo.

O gradiente descendente dispõe de alguns padrões de treinamento diferentes, como o *on-line*, ou estocástico, o em *mini-batches* e em *batches* (BROWNLEE, 2019). No treinamento *on-line*, cada dado de entrada acarretará na atualização de todos os pesos do modelo. No treinamento em *mini-batches*, os dados de entrada são divididos em lotes menores, que são usados para calcular o erro e atualizar o modelo; dessa forma, os pesos são atualizados apenas o processamento de um *mini-batch* por inteiro. Por fim, no treinamento em *batches*, a atualização dos pesos ocorre apenas após o processamento de todos os dados de entrada, utilizando do erro médio calculado em cada dado de entrada (KONEČNÝ et al., 2015).

3 Trabalhos Correlatos

Diversos trabalhos propuseram o uso de técnicas de análise de desempenho de algoritmos. Porém, muitos deles propõem o uso de técnicas puramente matemáticas que, dependendo do algoritmo, configuram um esforço bastante árduo ou até mesmo ineficaz em função do comportamento atípico do algoritmo. Para amenizar o enfoque matemático nas análises, algumas propostas sugerem o uso de métodos empíricos para a realização da análise de programas implementados.

3.1 Análise de desempenho de softwares baseada em métodos empíricos

Em (GOLDSMITH; AIKEN; WILKERSON, 2007) os autores propuseram um método empírico baseado na complexidade computacional para entender a escalabilidade de programas. Para este caso em particular, entende-se por escalabilidade a capacidade de ganho de desempenho do programa em função do incremento de novas cargas de trabalho (COULOURIS, 2012). O objetivo deste trabalho foi propor uma alternativa de análise do comportamento assintótico do programa sem o uso da abordagem matemática. A abordagem feita em (GOLDSMITH; AIKEN; WILKERSON, 2007) executa um determinado programa considerando diversas cargas de modo a obter e usar informações provenientes do programa em um modelo. Tal modelo prevê o desempenho do programa em função do tamanho da carga de trabalho. De posse do modelo, o intuito é comparar os resultados das simulações com as expectativas do programador ou com os limites teóricos assintóticos de modo a revelar *bugs* de desempenho ou confirmar que o desempenho do programa é escalonado conforme o desejado. A ferramenta usada para gerar e simular o modelo (*trend-prof*) é de propriedade dos autores.

Outra proposta interessante foi sugerida em (BOROVSKA; LAZAROVA, 2007). O método empírico proposto é baseado na comparação de desempenho de modelos de programação paralelos para avaliar o *speedup*, a eficiência e a escalabilidade usando algoritmos Minimax de busca paralela em árvores de jogos geradas no jogo *tic-tac-toe* (jogo da velha). O modelo computacional paralelo proposto pelos autores explora o particionamento da árvore em largura para cada nível da árvore de jogo e é baseado na combinação dos paradigmas algorítmicos paralelos “gestor-trabalhador” e “iterações assíncronas”. A comparação de desempenho foi feita para três modelos de programação paralelo (COULOURIS, 2012): *multithreaded*, *flat* (baseado na interface *Message Passing Interface* (MPI)) e híbrido (MPI+OpenMP). O *speedup* e a eficiência, bem como a escalabilidade (em relação

ao incremento ou alteração de recursos no computador) e seu impacto no desempenho do sistema paralelo foram estimados com base em resultados experimentais e comparados por análise de gráficos.

3.2 Análise de complexidade de Redes Neurais Artificiais utilizando abordagens puramente analíticas

Vários trabalhos utilizam técnicas analíticas para verificar a complexidade das RNA. São aplicadas técnicas mais tradicionais, focadas em métodos puramente matemáticos. Em (SERPEN; GAO, 2014), os autores analisaram a complexidade computacional e de comunicação de uma Perceptron Multicamadas implementada de forma distribuída e paralela em uma rede de sensores sem fio (RSSF).

Em (KON; PLASKOTA, 2000), os autores apresentam uma análise de complexidade de uma rede neural de três camadas, na qual cada neurônio pode ter funções de ativações distintas, que utiliza a função de base radial (RBF) na alimentação direta (*feed-forward*). Foram obtidos resultados inéditos para todas as classes de funções de ativação.

Ao utilizar abordagens puramente analíticas, torna-se necessária uma nova análise para cada novo tipo de arquitetura de algoritmo ou de rede neural. Se algumas características mudam na arquitetura da distribuição, novos resultados teóricos precisam ser produzidos. Em contrapartida, uma abordagem baseada em modelo visual e em simulações repetidas poderá ser generalizada para qualquer tipo de arquitetura de rede neural ou de distribuição no caso de algoritmos paralelos. Para cada algoritmo, um novo modelo precisará ser produzido, mas o princípio da análise permanecerá.

3.3 Redes de Petri para a modelagem de problemas

Na computação, diversos trabalhos vêm demonstrando a eficácia no uso das RdP para resolver seus problemas. Em (GUAN; LI; CHEUNG, 1996), os autores propuseram o uso de Redes de Petri Estocástica Generalizada (RdPEG) para modelar uma arquitetura composta de dois processadores paralelos, contendo cada um a sua própria memória local e ambos utilizando um barramento comum para acessar dois módulos de memória compartilhada.

Na área da Inteligência Artificial, também existem algumas propostas de uso das RdP, tais como: a associação das Redes de Petri Estocástica (RdPE) com uma Rede Neural Artificial (RNA) que utiliza um aprendizado supervisionado e não-supervisionado para a estimativa de eventos raros (SHEN; CHANG; JUANG, 2010), e a utilização da Lógica Nebulosa com uma RdP Distribuída para diagnosticar falhas em sistemas de potência

([YANG et al., 2010](#))

Em ([HANZÁLEK, 2003](#)), o autor utiliza as RdP para modelar uma RNA e analisa a complexidade do modelo utilizando definições formais, baseando-se na estrutura da RdP apresentada. Além disso, também são apresentadas técnicas baseadas na noção de componente conservativo de uma rede de Petri para a paralelização do processamento e para a simplificação do modelo apresentado.

3.4 Análise de desempenho de *softwares* baseada em ferramentas de modelagem visual

Alguns pesquisadores sugerem o uso de ferramentas de modelagem visual com o intuito de propor alternativas para o aprimoramento de sistemas computacionais sem focar em métodos matemáticos. Em ([FERREIRA; GOLDMAN; MONTEIRO, 2007](#)) a proposta se concentrou em usar grafos no desenvolvimento de protocolos de roteamento em redes dinâmicas de modo a propiciar menores rotas e maiores desempenhos no tráfego da rede.

Na abordagem sugerida por ([ZHAO et al., 2014](#)), os autores introduziram um método de modelagem de comportamento de *software* baseado em autômatos de estados finitos estendidos e em métodos de modelagem de comportamento de *software* existentes e focado em restrições sobre valores de dados e interações entre componentes de *software*. A modelagem de comportamento de *software* pode ser utilizada para detectar comportamentos anômalos, verificar protocolos, gerar casos de teste, capturar sequências de eventos inesperados e verificar a compatibilidade das propriedades do programa. Os autores combinaram as ferramentas Daikon e ESC/JAVA para obter as restrições e apresentá-las em arestas de autômatos de estados finitos estendidos para que a precisão do modelo construído pudesse ser melhorada. Os resultados das experiências mostraram que este modelo pode capturar muitas informações mais precisas e fornecer uma boa garantia para a análise, verificação e teste de *software*.

4 Modelo de simulação de uma Rede Neural Artificial

Como definido nos objetivos, foi construído um modelo de RdP no *CPN Tools* para que simulações pudessem ser realizadas sobre tal modelo. O modelo inicial foi baseado no modelo proposto por Hanzalek e em seguida cada transição foi hierarquizada, com o intuito de detalhar melhor o funcionamento do algoritmo e produzir resultados mais precisos decorrentes das simulações.

4.1 Modelo de Hanzalek

O modelo em RdPCH da Perceptron de Múltiplas Camadas construído durante o trabalho foi baseado no modelo proposto por Hanzalek em (HANZÁLEK, 2003), apresentado na Figura 10 e descrito pelas equações a seguir, que descrevem cada segmento do algoritmo; as mesmas equações foram utilizadas para a modelagem do modelo hierárquico, que será apresentado posteriormente.

As seguintes notações serão utilizadas nas equações:

- N_l é o número de neurônios na camada l ;
- k é um índice que representa cada iteração do algoritmo;
- $I_j^l(k)$ é o *input* do neurônio j da camada l ;
- $u_j^l(k)$ é o output do neurônio j da camada l ;
- $\delta_j^l(k)$ é o erro que será retropropagado pelo neurônio j da camada l ;
- $w_{ij}^l(k)$ é peso entre o neurônio i da camada $l - 1$ e o neurônio j da camada l ;
- η_l é a taxa de aprendizado;
- α_l é o fator de momentum na camada l ;

As equações apresentam o comportamento do algoritmo em cada etapa de sua execução: ativação, retropropagação e aprendizado.

- Equação de ativação:

$$I_j^l(k) = \sum_{i=1}^{N_{l-1}} [w_{ij}^l(k) \times u_i^{l-1}(k)] \quad (4.1)$$

$$\forall l = 1 \dots 3, \forall j = 1 \dots N_l$$

$$I_j^0(k) = j\text{-esimo input da rede neural} \quad (4.2)$$

$$u_j^l(k) = f(I_j^l(k)) = \frac{2}{1 + e^{-I_j^l(k)}} - 1 \quad (4.3)$$

$$\forall l = 0...3, \forall j = 1...N_l$$

- Equação de retropropagação (camada de *output*):

$$\delta_i^l(k) = f'(I_i^l(k)) \times (u_i^{\text{desired}}(k) - u_i^l(k)) \quad (4.4)$$

para $l = 3$

- Equação de retropropagação (camadas ocultas):

$$\delta_i^l(k) = f'(I_i^l(k)) \times \sum_{j=1}^{N_{l+1}} (\delta_j^{l+1}(k) \times w_i^{l+1} j(k)) \quad (4.5)$$

para $l = 2, 1$

- Equação de aprendizado

$$\Delta w_{ij}^l(k) = \eta^l \times \delta_j^l(k) \times u_i^{l-1}(k) + \alpha^l \times \Delta w_{ij}^l(k-1) \quad (4.6)$$

$\forall l = 1...3$

$$w_{ij}^l(k) = w_{ij}^l(k-1) + \Delta w_{ij}^l(k) \quad (4.7)$$

$\forall l = 1...3$

A rede proposta por Hanzalek representa o funcionamento de uma Perceptron com uma camada de entrada com dois neurônios, duas camadas ocultas com quatro neurônios e uma camada de saída com dois neurônios, como ilustrado na Figura 11.

Em seu modelo, Hanzalek representa os *inputs* da rede pelo lugar $P0$ e a aplicação da função sigmóide nos dados de *input* na transição $T0$ (equação 4.3).

Hanzalek também representa o processo de ativação, representados pelas transições $T1$, $T2$ e $T3$ na Figura 10, no qual para calcular o valor de *input* de um determinado neurônio em uma camada i , são usados o valores de *output* dos neurônios da camada $i - 1$ e os pesos que conectam cada um dos neurônios da camada $i - 1$ ao neurônio em questão, como definido na equação 4.1.

Os lugares $P1$, $P2$, $P3$ e $P4$ representam os *outputs* de todos os neurônios de cada camada, sendo $P1'$ a representação de uma cópia dos outputs em $P1$. $P2'$ e $P2''$ são representações de cópias dos *outputs* em $P2$. $P3'$ e $P3''$ são representações de cópias dos *outputs* em $P3$.

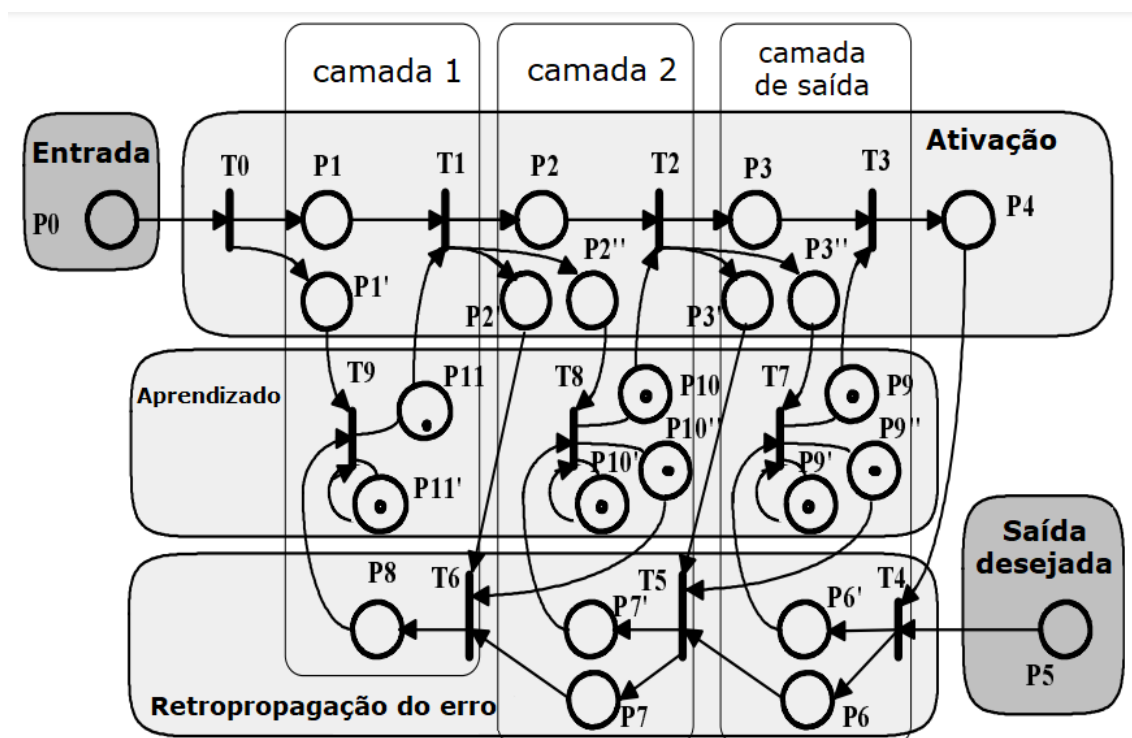


Figura 10 – Modelo em RdP para representação de um algoritmo de aprendizado em RN (HANZÁLEK, 2003)

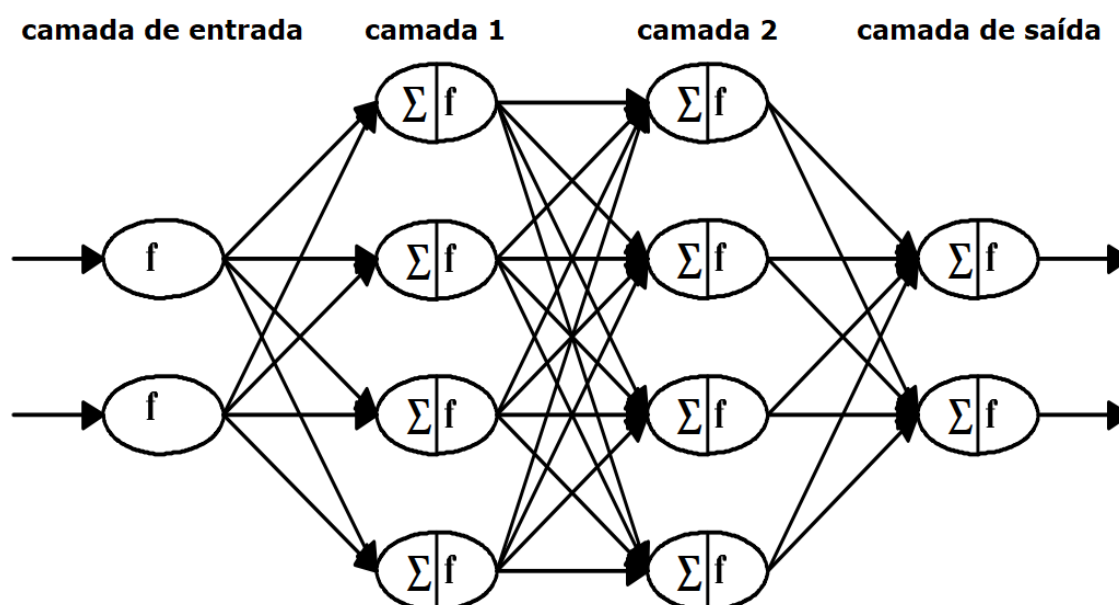


Figura 11 – Exemplo de uma rede neural multicamadas (2-4-4-2)

Os lugares $P11$, $P10$ e $P9$ representam, respectivamente, os pesos entre a camada de *input* e a primeira camada oculta, a primeira camada oculta e a segunda camada oculta, e a segunda camada oculta e a camada de *output*. $P11'$ representa uma cópia de $P11$. $P10'$ e $P10''$ representam cópias de $P10$. $P11'$ e $P11''$ representam cópias de $P11$.

Após a transição $T3$ ser disparada e aparecer uma ficha no lugar $P4$, a transição $T4$, que representa a avaliação do erro na camada de saída, é sensibilizada, utilizando a representação do *output* do lugar $P4$ e do *output* desejado representado pelo lugar $P5$ (equação 4.4). O erro calculado fica representado por fichas no lugar $P6$, e uma cópia desse erro no lugar $P6'$.

De acordo com a equação 4.5, utilizando o conceito de retropropagação, os erros das camadas ocultas (lugares $P7$ e $P8$) são calculados com base no erro das camadas subsequentes, além do peso que liga os neurônios das duas camadas e o *outputs* respectivos obtidos na fase de ativação.

Finalmente, utilizando os erros, os pesos da última iteração e os *outputs* de cada camada, são atualizados os pesos nas transições $T7$, $T8$ e $T9$ (equações 4.6 e 4.7). Apenas ao final do disparo de $T9$, um novo dado de entrada poderá ser processado, já que a transição $T1$ será sensibilizada novamente com a criação de uma ficha em $P11$.

Utilizando como base o modelo proposto por Hanzalek, a rede foi construída no ambiente do CPN Tools, como ilustrado na Figura 12. Diferentemente da RdP proposta por Hanzalek, foi decidido que para alcançar resultados mais precisos durante experimentações e testes, seria utilizado um modelo hierárquico, o qual permite expandir cada transição da rede original e detalhar as operações que na rede original estavam subentendidas.

Além da hierarquização, também foram adicionadas cores aos lugares, possibilitando a utilização de fichas com valores inteiros e de fichas com marcação de tempo. Um dos recursos oferecidos pelo CPN Tools que também foi empregado é lugar de fusão. Esse recurso permite a criação de vários lugares idênticos, vinculados entre si, representando um mesmo lugar, utilizado para reduzir a quantidade de arcos em uma mesma rede, facilitando a compreensão do modelo.

4.2 Modelo Hierárquico

Expandindo a transição $t0$ da Figura 12, temos a sub-rede ilustrada na Figura 13. Cada ficha no lugar de fusão *proc* representa um processador, cada um com um tempo associado, tempo esse que simboliza quando esse processador estará disponível para utilização.

A transição $t0$ na Figura 12, representa a ativação da função sigmóide na camada

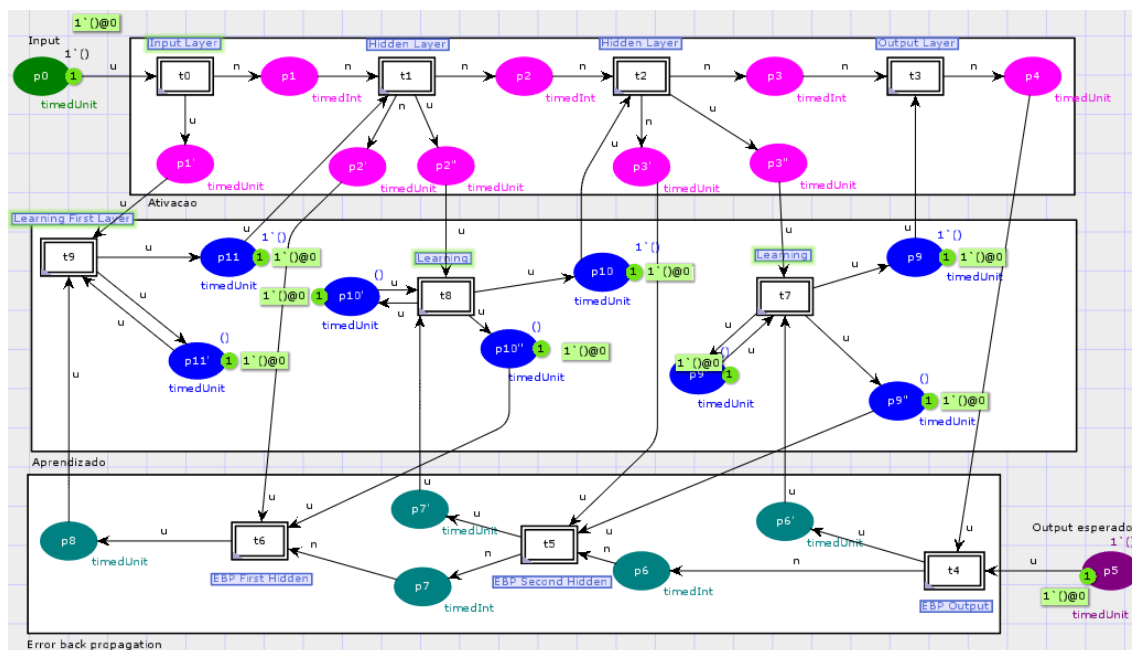
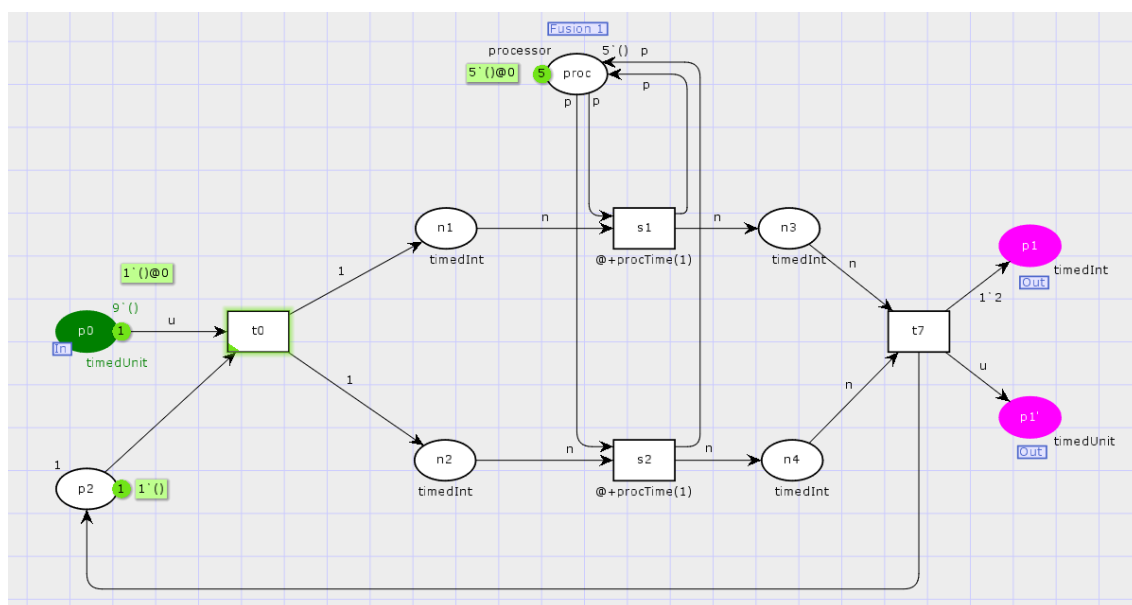
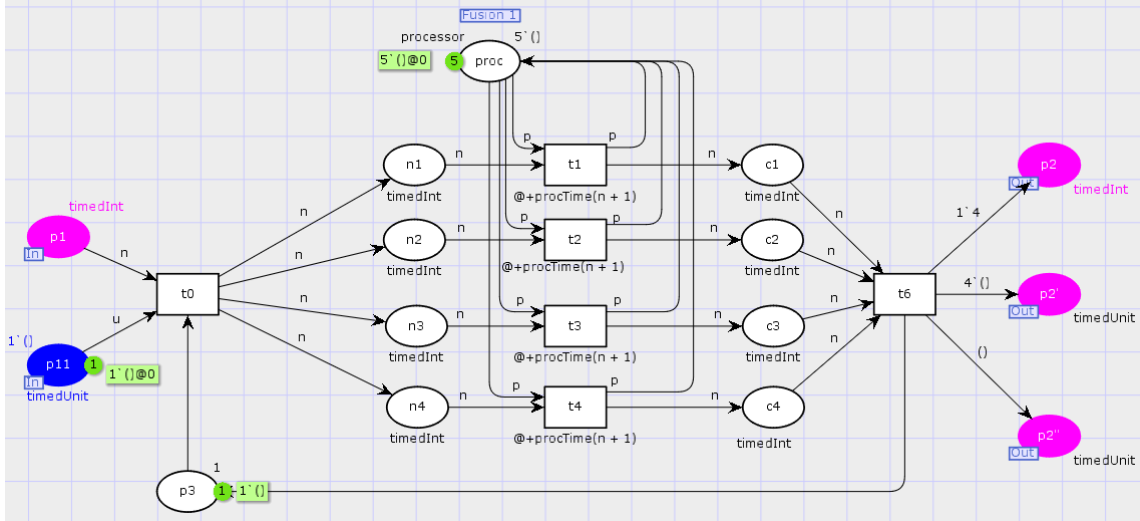


Figura 12 – RdPCH construída no CPN Tools

Figura 13 – Expansão da transição t_0 da Figura 12

Figura 14 – Expansão das transições $t1$ e $t2$ da Figura 12

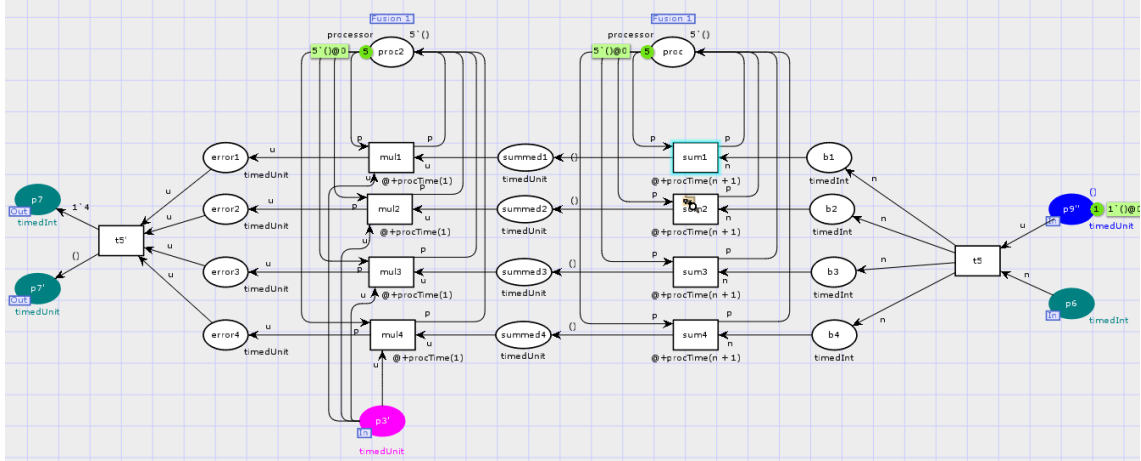
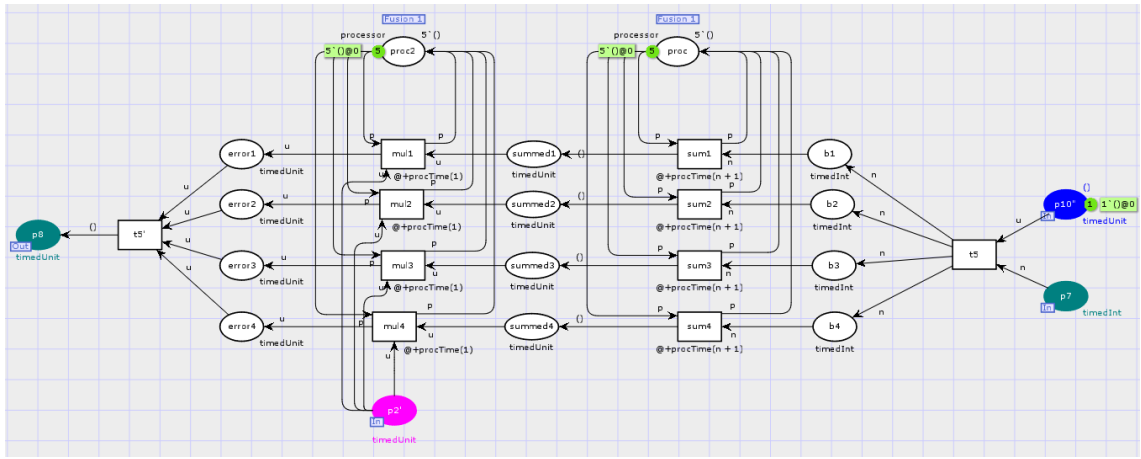
de entrada da rede. Na figura 13 é possível observar as transições $s1$ e $s2$, cada uma representando a ativação da função sigmóide em cada um dos neurônios da camada. Ao disparar essas transições, as fichas geradas em $n3$, $n4$ e $proc$ têm os tempos associados aumentados em uma unidade de tempo discreta ($+1$), fazendo com que elas só possam ser utilizadas em uma unidade de tempo depois na simulação. Leva-se em consideração que esse tempo discreto, para a análise de algoritmos, representa a execução de uma operação elementar, como geralmente é o caso em exemplos de análises de complexidade assintótica de algoritmos.

A inscrição no arco entre a transição $t7$ e o lugar $p1$ contém a informação de quantos neurônios existem na camada de *input*, informação que será necessária para calcular o tempo computacional das ativações das camadas subsequentes. O lugar $p2$ garante que as funções de ativação de um dado de entrada só sejam calculadas depois de todos os cálculos das funções de ativação, em todos os neurônios, do dado de entrada anterior.

A Figura 14 retrata a sub-rede que detalha as transições $t1$ e $t2$ da Figura 12, que representam os processos de ativação nas camadas ocultas da rede. Diferentemente da transição anterior, nas camadas ocultas, além do valor do *output* das camadas anteriores, também são utilizados os pesos entre cada neurônio da camada anterior e os da camada atual para o cálculo do input de cada neurônio, representados pelo lugar $p11$.

As transições $t1$, $t2$, $t3$ e $t4$ na Figura 14, representam as operações do cálculo do *input*, e do *output* de cada neurônio. Sendo n o número de *neurônios* da camada anterior, estão representadas $n + 1$ operações em cada uma dessas transições; n multiplicações para calcular o *input* e uma operação que representa o cálculo da função de ativação.

A transição $t3$ na Figura 12, detalhada na Figura 15, representa a ativação da última camada da rede. Uma diferença entre ela e as transições que representam os processos de ativação das camadas ocultas é que não é mais necessário enviar para a próxima

Figura 17 – Expansão da transição t_5 da Figura 12Figura 18 – Expansão da transição t_6 da Figura 12

$mul1$, $mul2$, $mul3$ e $mul4$, as multiplicações.

A Figura 18 mostra a transição t_6 da Figura 12, que representa o cálculo do erro na primeira camada oculta que será retropropagado. Essa transição se diferencia da transição t_6 apenas por não gerar uma cópia do seu erro calculado, já que não será mais retropropagado; apenas será utilizado para recalcular os pesos.

As transições t_7 e t_9 da Figura 12 são expandidas na Figura 19. Essas transições representam a parte de aprendizado do algoritmo, na qual são atualizados os pesos entre os neurônios de camadas adjacentes. Essa sub-rede possui uma transição para cada peso ($l1$ a $l8$) que liga os neurônios da segunda camada oculta à camada de *output* (t_7) e, analogamente, dos neurônios da camada de *input* aos da primeira camada oculta (t_9). Cada uma dessas transições, por sua vez, são expandidas na sub-rede ilustrada na Figura 20.

No modelo de Hanzalek, fica subentendido que ambos $\delta(k-1)$ e $w(k-1)$ estão representados em $p9'$. A transição *separate* representa a separação desses dados, que serão representados, respectivamente, por fichas em $\delta k-1$ e $w k-1$.

Utilizando como referência as equações apresentadas por Hanzalek, a transição *mul alpha, delta(k - 1)* representa a multiplicação entre α e $\delta(k - 1)$. O lugar *p6'* contém a representação do erro calculado na camada. A transição *mul lr, ebp* representa a multiplicação desse erro e do *learning rate*, de acordo com as equações, e a representação desse resultado é gerada no lugar *lr x ebp*. Esse resultado é utilizado em outra multiplicação com μ , representado pela transição *mul lr x ebp, mi* e seu respectivo resultado, gerado no lugar *lr x ebp x mi*. A transição *add* representa a primeira adição presente na equação de aprendizado apresentada por Hanzalek, e *add2*, a segunda adição.

Após as operações de adição indicadas pelas transições *add* e *add2* na Figura 20, a transição *join* é sensibilizada, representando a união dos dados $\delta(k)$ e $w(k)$ em uma estrutura de dados, dados esses que serão utilizados ao submeter um novo dado de entrada à rede neural.

A figura 21 mostra a expansão da transição *t8* da Figura 12. Diferentemente das transições *t7* e *t9*, há 16 transições, representando a atualização de cada peso existente entre as duas camadas ocultas, uma vez que cada uma possui 4 neurônios.

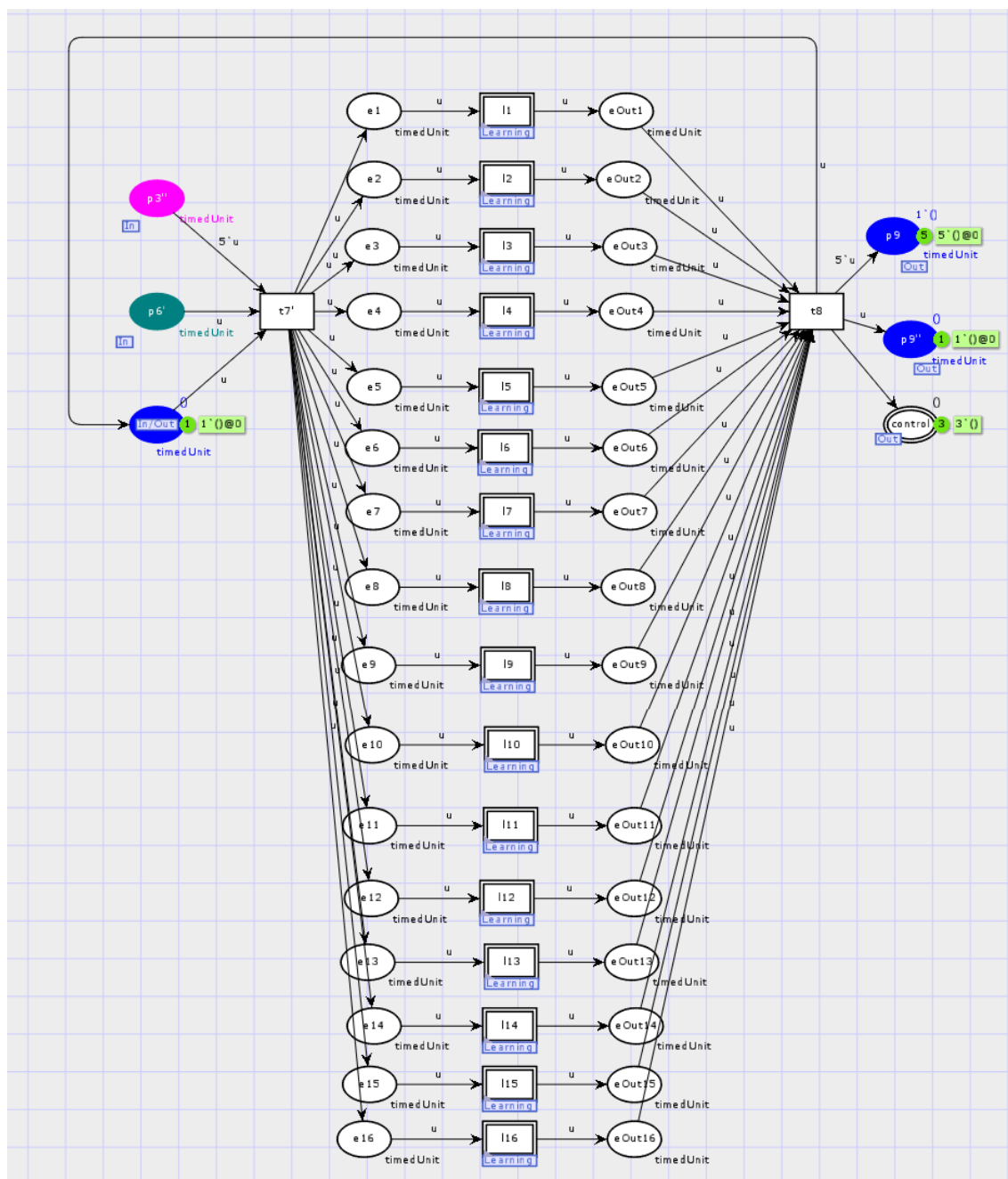
4.3 Adaptação do Modelo para Processamento em Lotes

A Figura 22 mostra a adaptação do modelo hierárquico construído no caso do funcionamento da Perceptron de Múltiplas Camadas considerando o processamento dos dados de entrada em *mini-batches* ou em *batches*, caso particular no qual os *mini-batches* têm tamanho igual ao número de *inputs*.

Como explicado em (KONEČNÝ et al., 2015) e (BROWNLEE, 2019), diferentemente do modelo que processa os dados de forma sequencial, no processamento em *mini-batches*, diferentes entradas de um mesmo lote são propagadas pela rede utilizando os mesmos pesos da rede. Apenas após os cálculos de todos os erros dos *inputs* de um mesmo lote, os mesmos serão retropropagados e os pesos atualizados.

No exemplo da Figura 22, que representa a PMC com processamento em *mini-batches*, com lotes de tamanho 5, é possível observar a adição dos lugares *control* e *control2* e da transição *tcontrol*. Fichas no lugar *control* são geradas após o término da atualização dos pesos entre duas camadas. Assim que todos os pesos forem atualizados, haverão 3 fichas no lugar, sensibilizando a transição *tcontrol*. Após o disparo de *tcontrol*, serão geradas n fichas em *control2*, sendo n o tamanho do *mini-batch*, 5 na Figura 22. As fichas em *control2* serão consumidas, uma a uma, a cada disparo de *t0*, e ao consumir todas as fichas de *control2*, novos disparos de *t0* só serão possíveis após um novo disparo de *tcontrol*, no próximo lote.

Além dos lugares e transição de controle, o peso de alguns arcos foram modificados,

Figura 21 – Expansão da transição t_8 da Figura 12

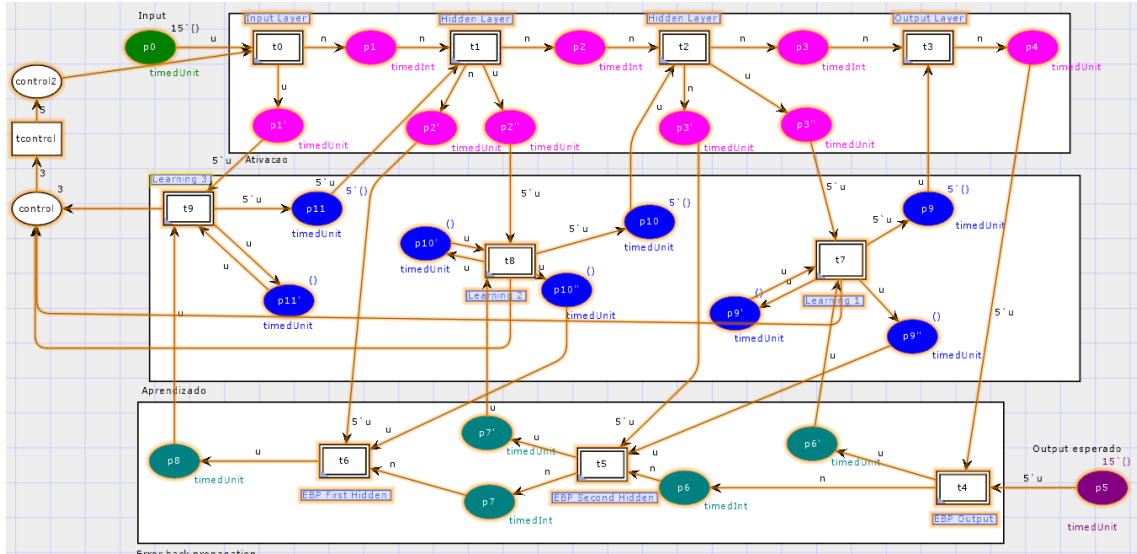


Figura 22 – RdPCH que representa o uma PMC realizando processamento em mini-batches

passando de uma ficha necessária para a sensibilização de uma transição para o tamanho do *mini-batch*, 5 no exemplo. O arco que liga o lugar $p4$ à transição $t4$, por exemplo, foi inscrito com o valor 5, significando que a transição $t4$ só poderá ser disparada quando os *outputs* das 5 entradas do lote forem produzidos.

4.4 Modelo de Alocação de Recursos

A alocação de recursos na rede é representada pelo lugar de fusão *proc*. Esse lugar, utilizando a ferramenta de lugar de fusão, está conectado a todas as transições que representam operações elementares na rede. Cada ficha no lugar *proc* representa um processador e tem um valor de tempo associado, esse valor representa em que momento na simulação essa ficha poderá ser consumida para o disparo de uma transição.

Para simular uma quantidade infinita de processadores, só é necessário remover o lugar *proc* ou os arcos que o liga a todas as transições que representam operações elementares. Dessa forma, não há nenhuma concorrência por recurso e as transições podem ser disparadas assim que forem sensibilizadas.

5 Simulação dos Modelos em Rede de Petri da RNA

O objetivo da análise das simulações dos modelos produzidos está relacionado com os tempos de execução dos modelos em conjunto com uma quantidade de processadores alocados para a tarefa. Dessa forma, será possível calcular o *speedup* e encontrar o ponto de saturação de processadores, ou seja, um número máximo de processadores que contribuem para a atenuação dos tempos de execução.

O *speedup* é definido como a relação entre o tempo gasto para executar uma tarefa com um único processador e o tempo gasto com N processadores. Dessa forma, o speedup representa uma medida de ganho em tempo.

Com esse objetivo e para fins de comparação, foram realizadas simulações para os casos de processamento *on-line* com 15 dados de entrada, processamento em *mini-batches* com 15 dados de entrada e lotes de tamanho 5 e processamento de *batches* com 15 dados de entrada e *batches* de tamanho 15.

Como mencionado anteriormente, cada operação que envolve o processador tem uma unidade de tempo 1 associado, impedindo que a ficha que representa o processador seja utilizada novamente em outras operações até por uma unidade de tempo, que corresponde a uma operação elementar.

As simulações foram feitas utilizando as ferramentas disponibilizadas pelo *CPN Tools*, incluindo monitores, que foram empregados nas transições para identificação de gargalos no modelo nos caso com múltiplos processadores.

5.1 Simulação do Processamento *On-line*

A primeira simulação realizada foi referente ao modelo em RdPCH da RNA com um processamento puramente sequencial, no qual os pesos são atualizados a cada dado de entrada que passa pela rede.

Primeiramente, foi executada uma simulação com uma infinidade de processadores para estabelecer um limite superior de tempo para a simulação. Para tal, os arcos que ligam o lugar *proc* às transições foram retirados. Dessa forma, as transições não dependem mais dos processadores para serem disparadas.

Com a informação do limitante superior para número de processadores, foram realizadas simulações do modelo, aumentando a cada simulação o número de processadores em 1, até atingir o tempo mínimo determinado pela simulação feita com uma infinidade

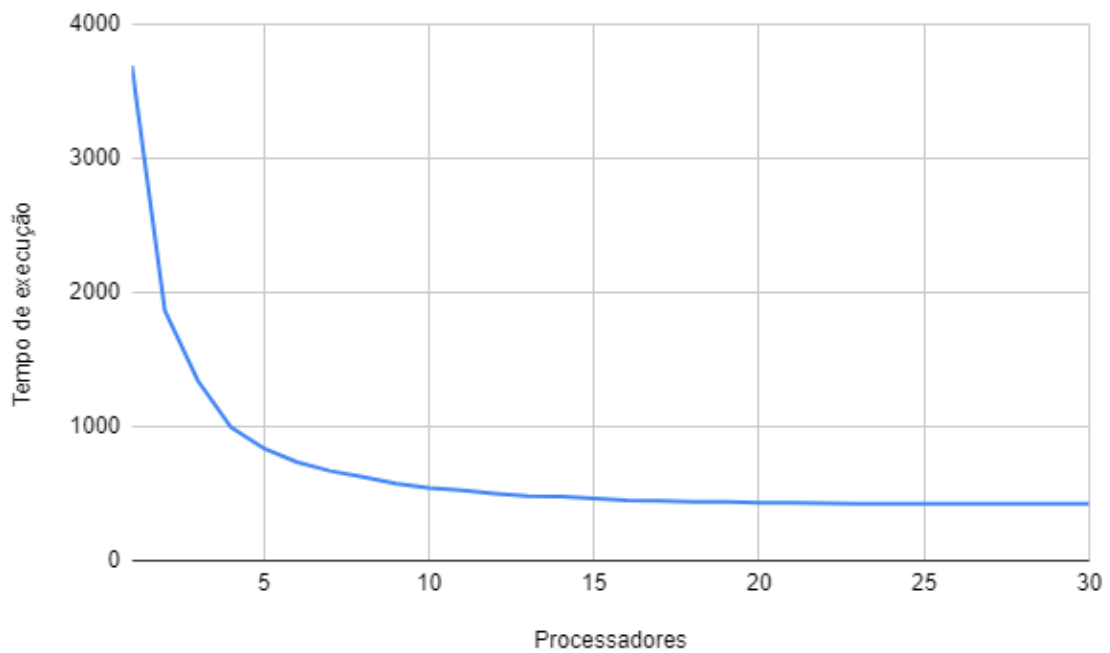


Figura 23 – Gráfico de simulação do modelo com processamento on-line

de processadores. O resultado dessas simulações é apresentado na Figura 23.

A partir do gráfico obtido, é possível observar que o tempo de simulação diminui consideravelmente com o aumento do número de processadores. Isso acontece porque mais operações podem ser realizadas em paralelo ao aumentar o número de processadores.

O limitante superior para o número de processadores é 30, ou seja, a partir de 30 processadores, adicionar mais processadores não acarretará na redução do tempo de simulação. Com um processador, o tempo de simulação é de 3690 unidades de tempo e com 30 processadores, o tempo de simulação é de 422 unidades de tempo.

Além disso, fica claro através do gráfico que a diminuição do tempo de simulação é mais acentuada quando há poucos processadores e mais atenuada a partir de 5 processadores. Há, por exemplo, uma queda de aproximadamente 77% do tempo de simulação (2854 unidades de tempo) de 1 para 5 processadores e apenas 50% de redução no tempo de simulação (414 unidades de tempo) de 5 para 30 processadores.

O gráfico apresentado na Figura 24 mostra o *speedup* calculado para N processadores. Com 2 processadores, o *speedup* obtido foi de aproximadamente 1,97, significando que o tempo de execução com dois processadores foi 1,97 vezes mais rápido do que o tempo de execução com apenas um processador.

Com 30 processadores, o *speedup* obtido foi de 8,74. Isso significa que o maior ganho possível de velocidade em tempo de execução seria 8,74 vezes em relação às execuções com apenas um processador, já que 30 é o limitante superior para número de processadores que diminuiria o tempo de execução.

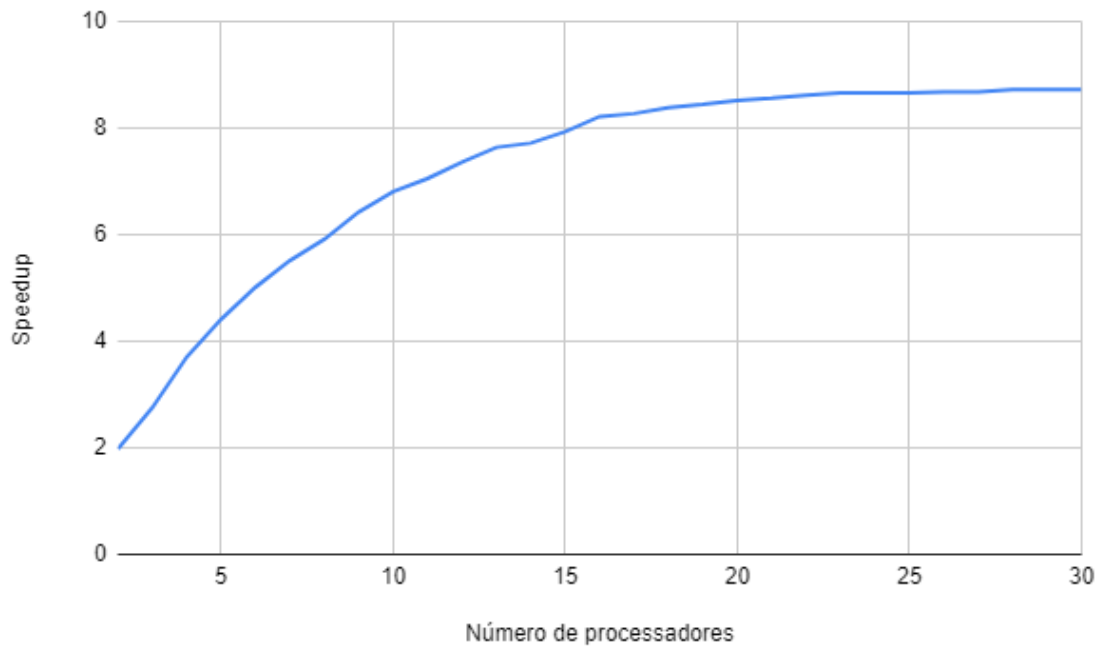


Figura 24 – Gráfico que mostra o speedup do modelo utilizando do processamento *on-line* com N processadores

5.2 Simulação do Processamento da RNA em *Mini-batches*

Para a segunda simulação que foi executada, foi utilizado o modelo adaptado para o processamento em *mini-batches*. Assim como no modelo utilizado na primeira simulação, os tempos de execução que foram obtidos são referentes ao processamento de 15 dados de entrada pela rede. No entanto, como trata-se do processamento em *mini-batches*, os 15 dados de entrada foram divididos em lotes de 5, atualizando os pesos da rede apenas quando todos os dados de um mesmo lote forem processados.

Da mesma forma como nas simulações do processamento *on-line*, primeiro foi determinado o limitante superior para o número de processadores ao realizar uma simulação desconsiderando os arcos dos lugares que representam os processadores. Em seguida, uma sequência de simulações foi realizada, iniciando a primeira com 1 processador e a cada nova simulação, aumentando o número de processadores em 1 até atingir o tempo mínimo de simulação alcançável.

O gráfico apresentado na figura 25 apresenta os resultados do experimento. Uma vez que os pesos do modelo são atualizados apenas quando todos os dados de entrada de um mesmo lote chegam ao final da rede, era de se esperar uma redução do tempo de simulação no geral, uma vez que o processo da retropropagação e atualização dos pesos acontece 5 vezes menos, já que os lotes têm tamanho 5.

No caso da simulação em *mini-batches*, foi obtido um limitante superior para o número de processadores de 20. O tempo de simulação para 1 processador foi de 1266

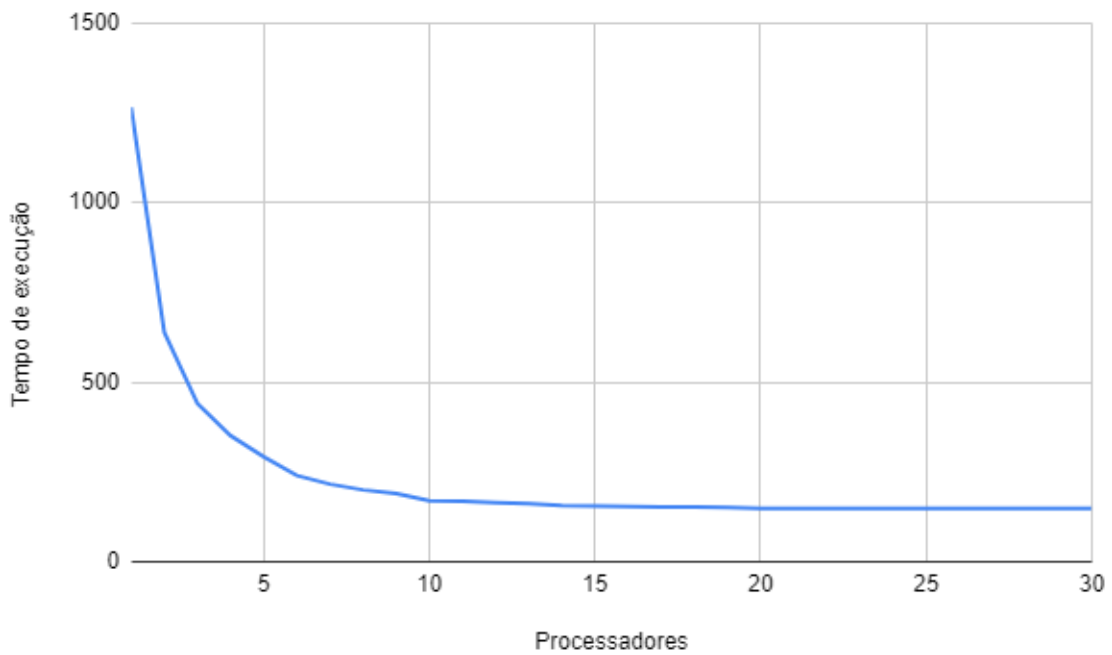


Figura 25 – Gráfico de simulação do modelo com processamento em mini-batches

unidades de tempo, redução de 2424 unidades de tempo (65,7%) em comparação com tempo no *processamento on-line* para 1 processador. Com 20 processadores, o tempo medido foi de 148 unidades de tempo, uma redução de 1118 unidades de tempo (88%) em comparação com o processamento em *mini-batches*, 1 processador, e 285 unidades de tempo (66%) em relação ao processamento *on-line* com 20 processadores.

Assim como no processamento *on-line*, observa-se que quanto mais processadores são adicionados, menos o tempo de simulação é reduzido, formando curvas muito semelhantes.

Na Figura 26, é possível visualizar o gráfico exibindo os resultados alcançados do valor do *speedup* em relação ao número de processadores. Com 2 processadores, a simulação foi cerca de 1,98 vezes mais rápida e com 20 processadores, o limitante para o modelo construído com processamento em *mini-batches*, 8,55 vezes mais rápida.

5.3 Simulação do Processamento em *Batches*

No último grupo de simulações, foi utilizado o modelo com processamento em *batches*, no qual os pesos do modelo são atualizados apenas ao final de cada época, ou seja, após todos os dados de entrada passarem pela rede. Assim como nos dois outros casos, os resultados obtidos são referentes a simulações com 15 dados de entrada.

Da mesma maneira que nos casos dos processamentos *on-line* e em *mini-batches*, a princípio foi estabelecido um limitante superior para o número de processadores que

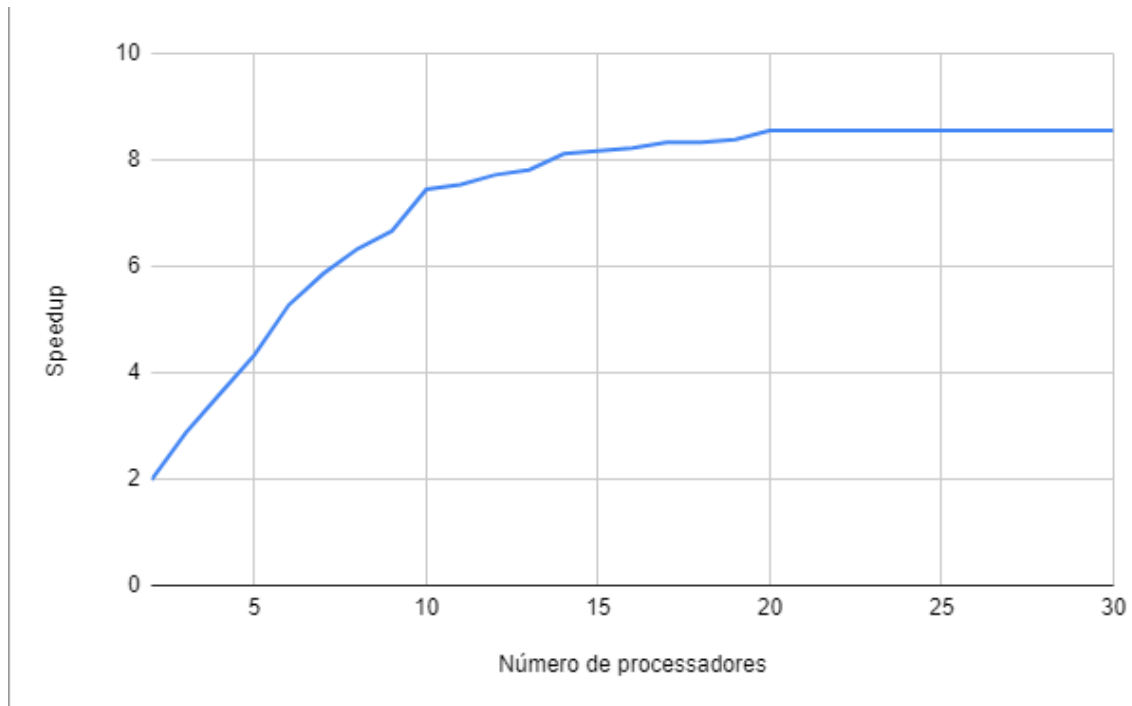


Figura 26 – Gráfico que mostra o speedup do modelo utilizando do processamento em *Mini-batches* com N processadores

contribuiriam para um decréscimo no tempo de simulação. Em seguida, uma série de simulações foi realizada, incrementando a cada simulação o número de processadores no modelo, começando por 1 e parando no limite determinado anteriormente.

Os resultados das simulações podem ser observados no gráfico apresentado na Figura 27. Novamente, assim como no caso do processamento em *mini-batches*, era esperada uma diminuição nos tempos de simulação, já que a retropropagação e a atualização dos pesos acontece apenas uma vez por época de treinamento, 15 vezes menos que no processamento *on-line* e 3 vezes menos que no processamento em *mini-batches* com lotes de tamanho 5.

O limitante superior encontrado para o número de processadores foi 20. Com 1 processador, o tempo de simulação obtido foi 862 unidades de tempo, 77% menor que no processamento *on-line* e 32% menor que no processamento em *mini-batches*. Com 20 processadores, o tempo obtido foi 100 unidades de tempo, uma redução de 762 unidades de tempo (88%) em relação à simulação com 1 processador. Comparando com as outras formas de processamento, a redução foi de 77% e 32% para os processamentos *on-line* e em *mini-batches*, respectivamente.

O gráfico apresentado na Figura 28 exibe os valores de *speedup* calculados para um número N de processadores. Com 2 processadores, o modelo em *batches* apresentou um speedup de 1,98. Já com 20, o speedup obtido foi de 8,62.

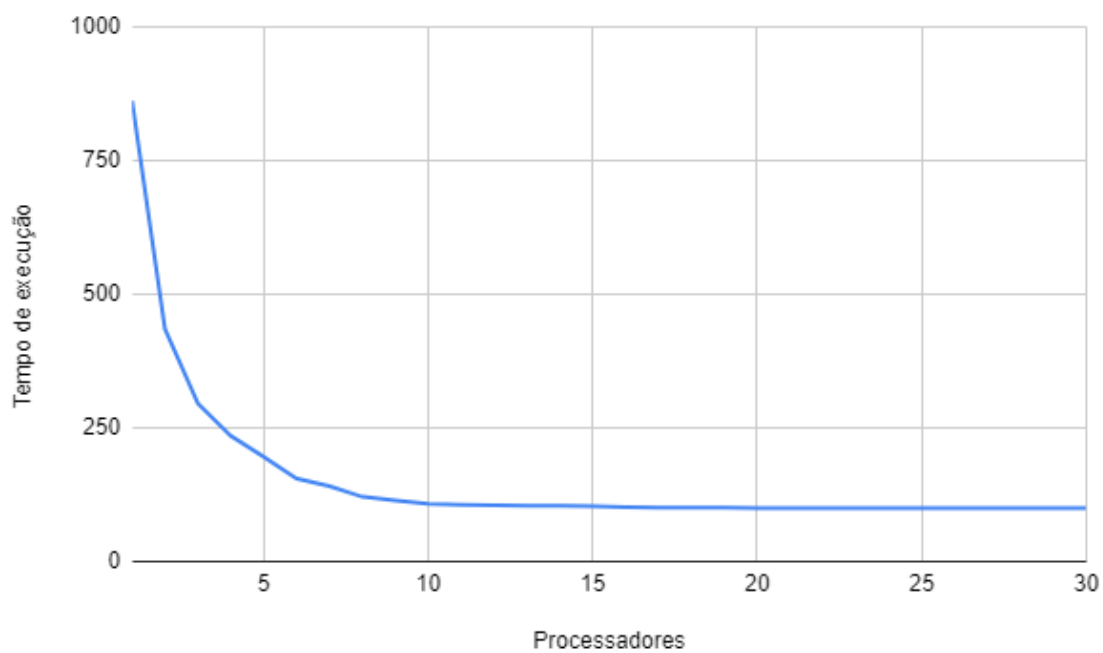
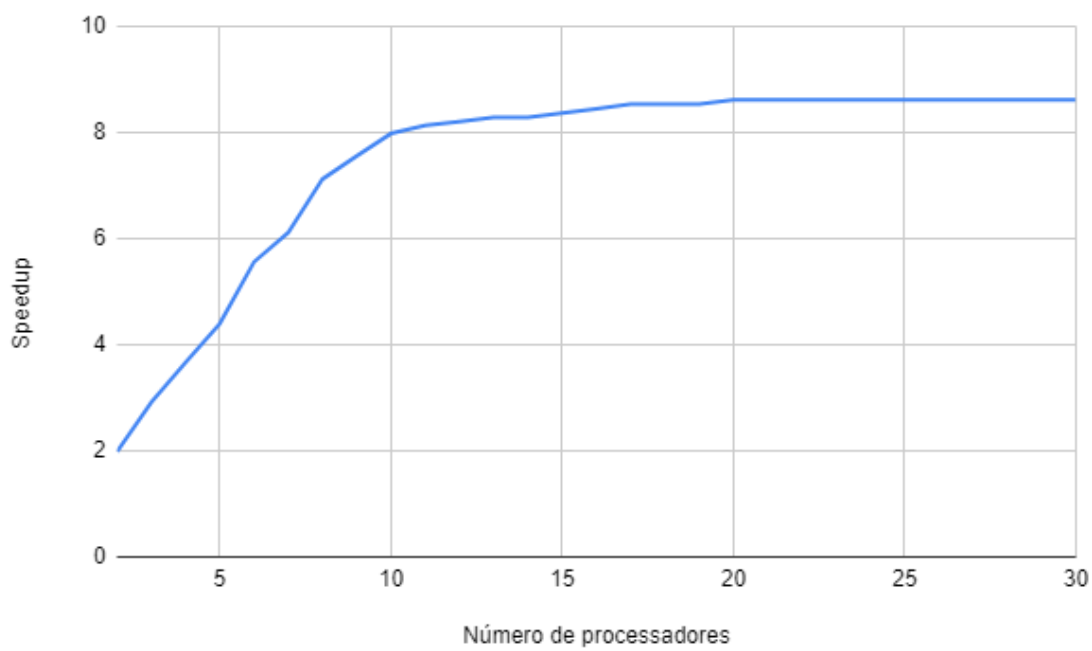


Figura 27 – Gráfico de simulação do modelo com processamento em batches

Figura 28 – Gráfico que mostra o speedup do modelo utilizando o processamento em *batches* com N processadores

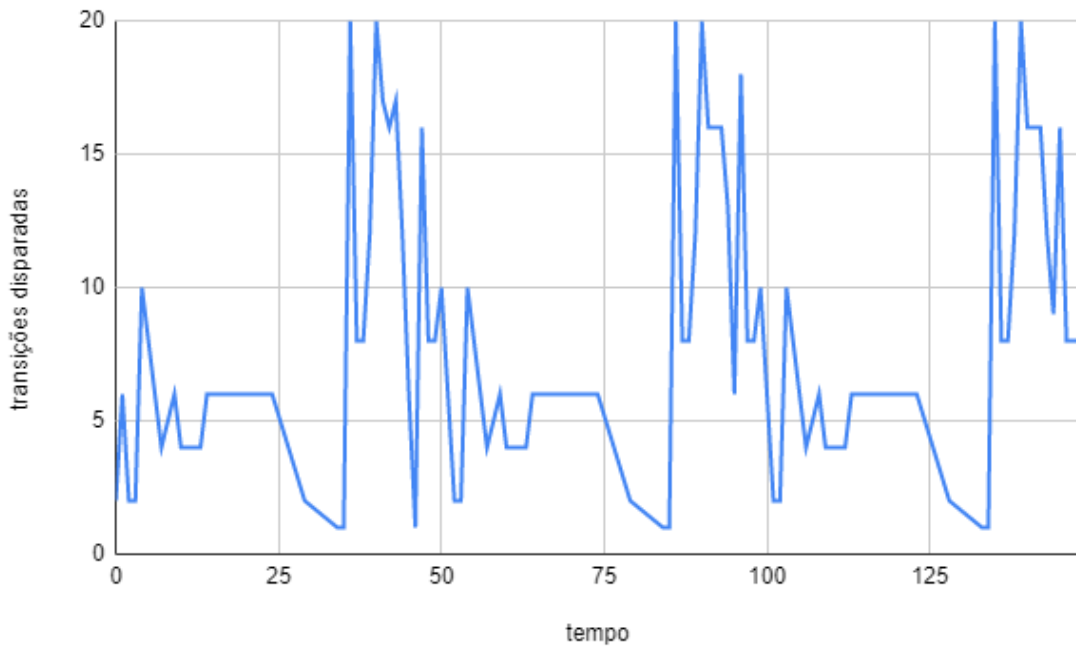


Figura 29 – Gráfico apresentando a quantidade de transições disparadas a cada momento da simulação

5.4 Análise dos Monitores

É possível fazer uma análise mais detalhada ao empregar o uso dos monitores disponibilizados pelo *CPN Tools*. A ferramenta pode ser aplicada em uma transição específica, fazendo com que um histórico de disparos da transição seja gerado após o término da simulação.

Os monitores foram aplicados em todas as transições do modelo com processamento em *mini-batches*; os resultados estão representados no gráfico apresentado na Figura 29.

Através do gráfico, é possível observar certa periodicidade, decorrente do fato de que no caso do processamento em *mini-batches* com 15 dados de entrada e 5 como tamanho de lote, o ciclo de *forward propagation* e retropropagação acontece três vezes.

Os momentos da simulação com maior número de transições sendo disparadas ao mesmo tempo são os instantes 36, 40, 86, 90, 135 e 139, todos com 20 transições disparadas ao mesmo tempo. Ao analisar esses instantes em específico, percebe-se que nesses momentos as transições que estão sendo disparadas são referentes às partes que representam as somas realizadas na retropropagação (transições *sum1*, *sum2*, *sum3* e *sum4* da Figura 18) e às multiplicações entre a taxa de aprendizagem e erro retropropagado (transição *mul lr, ebp* da Figura 20) e o fator de momentum e $\delta(k-1)$ (transição *mul alpha, delta(k-1)* da Figura 20). Logo, é possível concluir que são essas as operações que mais se beneficiariam de um número elevado de processadores (paralelização das operações

correspondentes).

Por outro lado, ao analisar os instantes com menos disparos de transição, os instantes 34, 35, 84, 85, 133 e 134, cada um com 1 transição disparada, fazem referência a duas transições em específico; a transição *sub* da Figura 16 nos momentos 34, 84 e 133 e a transição *mul* da Figura 16 nos momentos 35, 85 e 134.

Essas transições, de fato, não deveriam ser disparadas simultaneamente com outras transições, já que são referentes ao cálculo do erro na camada de saída. Para que essa transição seja sensibilizada, todos os dados de entrada já devem ter passado por toda a rede; logo nenhuma transição anterior às transições em questão pode ser disparada ao mesmo tempo. Como todas as transições subsequentes dependem do resultado do erro calculado na camada de saída, também não podem ser disparadas no mesmo instante.

Tais resultados poderão ser considerados numa proposta de projeto de arquitetura distribuídas para explorar as partes da RNA que podem ser processadas em paralelo. Em particular, os resultados obtidos através do uso de monitores indicam as partes do modelo da RNA que poderão se aproveitar melhor da disponibilidades de diversos processadores.

6 Considerações Finais

Este trabalho, inicialmente, formalizou e implementou o modelo de uma Perceptron de Múltiplas Camada (2-4-4-2) proposto por Hanzalek em ([HANZÁLEK, 2003](#)) utilizando de Redes de Petri Coloridas no ambiente software CPN Tools. O CPN Tools fornece diversas ferramentas que foram utilizadas para a simulação de modelagem da RdP, como lugares de fusão e a possibilidade de introduzir hierarquização ao modelo.

Com a adição de um modelo hierárquico foi possível particularizar cada transição do modelo, definindo, de acordo com as equações fornecidas por Hanzalek, as operações realizadas em cada momento do algoritmo, possibilitando um aperfeiçoamento na precisão do modelo e acarretando em medições de tempo mais precisas durante as simulações.

Para facilitar a compreensão do modelo, a ferramenta de lugar de fusão foi aplicada à rede. Com a utilização desse recurso, o único lugar que representava a quantidade de processadores disponíveis para a utilização foi separado em vários outros lugares, representando ainda o mesmo lugar. As fichas em todos os lugares de um mesmo lugar de fusão estão sempre sincronizadas entre si. Dessa forma, a quantidade de arcos que antes saía do mesmo lugar foi reduzida, deixando o modelo mais legível e organizado.

Além da modelagem, foram estudados os custos computacionais, de acordo com o número de operações de cada transição do modelo para ampliar a efetividade e precisão dos testes e simulações que serão realizados.

Com o modelo pronto, foi possível realizar as simulações para que pudesse ser feito os cálculos do *speedup*. O modelo foi simulado no ambiente do *CPN Tools* com diferentes números de processadores e com três tipos de processamentos diferentes, *on-line*, em *mini-batches* e em *batches*. Foi possível observar uma aceleração no tempo de simulação já esperada conforme o número de processadores aumentava. Além disso, como já esperado, os tempos das simulações em *batches* foram os mais rápidos, seguidos pelos processamentos em *mini-batches* e *on-line*, respectivamente.

Utilizando os monitores de transição do *CPN Tools*, foram identificadas as partes do modelo com maior e menor potenciais de paralelização ao analisar os momentos em que cada transição foi disparada durante a simulação.

Finalmente, após obter os tempos de simulação com um e mais processadores, foi possível calcular o *speedup* para cada número de processadores sem a necessidade de implementar o algoritmo em si como feito em ([BOROVSKA; LAZAROVA, 2007](#)).

Uma possível abordagem futura seria a utilização de técnicas de *process mining* ([SAHU; CHAKRABORTY; NAYAK, 2018](#)) para a detecção de partes que poderiam ser

paralelizadas no modelo através de *event logs*, como os que foram gerados utilizando os monitores disponibilizados pelo *CPN Tools*.

O resultado das técnicas de *Process Mining* poderiam em particular indicar automaticamente, baseando-se exclusivamente do levantamento dos dados fornecidos pelos monitores associados à diversas transições do modelo, os processos paralelos que poderiam se beneficiará de um projeto de arquitetura distribuída.

Também, será interessante aplicar a técnica proposta neste trabalho às arquiteturas diversas de RNA que são utilizadas nos diversos problemas cujas técnicas de resolução se apoiam principalmente nas técnicas de deep learning e que necessitam tempos extensos de processamento das RNA envolvidas.

Referências

- AALST, W. Van der; STAHL, C. *Modeling business processes: a petri net-oriented approach*. [S.l.]: MIT press, 2011. Citado na página 16.
- AMARI, S. et al. *The handbook of brain theory and neural networks*. [S.l.]: MIT press, 2003. Citado na página 20.
- BARBOSA, V. C. *An Introduction to Distributed Algorithms*. Cambridge, Massachusetts: The MIT Press, 1996. Citado na página 9.
- BISHOP, C. M. et al. *Neural networks for pattern recognition*. [S.l.]: Oxford university press, 1995. Citado na página 21.
- BOROVSKA, P.; LAZAROVA, M. Efficiency of parallel minimax algorithm for game tree search. In: *Proceedings of the 2007 international conference on Computer systems and technologies*. [S.l.: s.n.], 2007. p. 1–6. Citado 3 vezes nas páginas 9, 23 e 46.
- BROWNLEE, J. *A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size*. 2019. Disponível em: <<https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/>>. Citado 2 vezes nas páginas 22 e 35.
- CARDOSO, J.; VALETTE, R. *Redes de petri*. [S.l.]: Editora da UFSC Florianópolis, 1997. Citado 3 vezes nas páginas 6, 12 e 13.
- CORMEN, T. H. *Introduction to Algorithms*. Cambridge, Massachusetts: The MIT Press, 2009. Citado na página 9.
- COULOURIS, G. F. *Distributed systems: concepts and design/George Coulouris...[et al.]*. [S.l.]: Boston: Addison-Wesley,, 2012. Citado 2 vezes nas páginas 9 e 23.
- FERNEDA, E. Redes neurais e sua aplicação em sistemas de recuperação de informação. *Ciência da Informação*, SciELO Brasil, v. 35, n. 1, p. 25–30, 2006. Citado na página 21.
- FERREIRA, A.; GOLDMAN, A.; MONTEIRO, J. On the evaluation of shortest journeys in dynamic networks. In: IEEE. *Sixth IEEE International Symposium on Network Computing and Applications (NCA 2007)*. [S.l.], 2007. p. 3–10. Citado na página 25.
- FERREIRA, L. et al. Utilização de redes neurais artificiais como estratégia de previsão de preços no contexto de agronegócio. *RAI-Revista de Administração e Inovação*, Universidade de São Paulo, v. 8, n. 4, p. 6–26, 2011. Citado na página 21.
- GOLDSMITH, S. F.; AIKEN, A. S.; WILKERSON, D. S. Measuring empirical computational complexity. In: *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. [S.l.: s.n.], 2007. p. 395–404. Citado 2 vezes nas páginas 9 e 23.

- GUAN, H.; LI, C.-K.; CHEUNG, T.-Y. A generalized stochastic petri net approach for modeling multiprocessor parallel processing systems. In: IEEE. *Proceedings of 1996 IEEE Second International Conference on Algorithms and Architectures for Parallel Processing, ICA/sup 3/PP'96*. [S.l.], 1996. p. 413–420. Citado na página 24.
- HANZÁLEK, Z. *Parallel Algorithms for Distributed Control-A Petri Net Based Approach*. Tese (Doutorado) — Citeseer, 2003. Citado 6 vezes nas páginas 6, 10, 25, 26, 28 e 46.
- HASSOUN, M. H. et al. *Fundamentals of artificial neural networks*. [S.l.]: MIT press, 1995. Citado na página 10.
- HAYKIN, S. *Neural networks: a comprehensive foundation*. [S.l.]: Prentice-Hall, Inc., 2007. Citado na página 20.
- HRISTEV, R. *The ANN book*. [S.l.]: Edition, 1998. Citado 2 vezes nas páginas 6 e 20.
- JENSEN, K. et al. *CPN Tools*. 2018. Disponível em: <<http://cpntools.org>>. Citado 2 vezes nas páginas 10 e 16.
- JENSEN, K.; KRISTENSEN, L. M. *Coloured Petri nets: modelling and validation of concurrent systems*. [S.l.]: Springer Science & Business Media, 2009. Citado 3 vezes nas páginas 10, 15 e 16.
- JÚNIOR, C. M. M.; JULIA, R. M. S.; JULIA, S. Modeling recursive search algorithms by means of hierarchical colored petri nets and cpn tools. In: IEEE. *2015 12th International Conference on Information Technology-New Generations*. [S.l.], 2015. p. 788–791. Citado na página 10.
- JÚNIOR, C. M. M. et al. A new approach to evaluate the complexity function of algorithms based on simulations of hierarchical colored petri net models. In: *Information Technology-New Generations*. [S.l.]: Springer, 2018. p. 555–564. Citado na página 10.
- KON, M. A.; PLASKOTA, L. Information complexity of neural networks. *Neural Networks*, Elsevier, v. 13, n. 3, p. 365–375, 2000. Citado na página 24.
- KONEČNÝ, J. et al. Mini-batch semi-stochastic gradient descent in the proximal setting. *IEEE Journal of Selected Topics in Signal Processing*, IEEE, v. 10, n. 2, p. 242–255, 2015. Citado 2 vezes nas páginas 22 e 35.
- KRISTENSEN, L. M.; CHRISTENSEN, S.; JENSEN, K. The practitioner's guide to coloured petri nets. *International Journal on software Tools for technology Transfer*, Springer-Verlag, v. 2, n. 2, p. 98–132, 1998. Citado na página 18.
- MURATA, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, IEEE, v. 77, n. 4, p. 541–580, 1989. Citado 2 vezes nas páginas 12 e 13.
- RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *nature*, Nature Publishing Group, v. 323, n. 6088, p. 533–536, 1986. Citado na página 22.
- SAHU, M.; CHAKRABORTY, R.; NAYAK, G. A task-level parallelism approach for process discovery. *International Journal of Engineering & Technology*, v. 7, n. 4, p. 2446–52, 2018. Citado na página 46.

- SERPEN, G.; GAO, Z. Complexity analysis of multilayer perceptron neural network embedded into a wireless sensor network. *Procedia Computer Science*, Elsevier, v. 36, p. 192–197, 2014. Citado na página [24](#).
- SHEN, V. R.; CHANG, Y.-S.; JUANG, T. T.-Y. Supervised and unsupervised learning by using petri nets. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, IEEE, v. 40, n. 2, p. 363–375, 2010. Citado na página [24](#).
- SILVA, I. N. D.; SPATTI, D. H.; FLAUZINO, R. A. Redes neurais artificiais para engenharia e ciências aplicadas curso prático. *São Paulo: Artliber*, 2010. Citado na página [21](#).
- TOMAZ, L. B. P. et al. Adaba: uma nova abordagem de distribuição do alfa-beta-aplicação ao domínio do jogo de damas. Universidade Federal de Uberlândia, 2018. Citado 2 vezes nas páginas [6](#) e [21](#).
- YANG, J.-w. et al. Notice of retraction: A distributed fault diagnosis approach in power system based on fuzzy reasoning petri net. In: IEEE. *2010 Asia-Pacific Power and Energy Engineering Conference*. [S.l.], 2010. p. 1–4. Citado na página [25](#).
- ZHAO, X. et al. Research on software behavior modeling based on extended finite state automata. IET, 2014. Citado na página [25](#).