CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS TIMÓTEO

Vinícius de Moraes Ribeiro

ESTIMATIVA DE VITÓRIA EM DOTA 2 UTILIZANDO REDES NEURAIS ARTIFICIAIS

Timóteo

2018

Vinícius de Moraes Ribeiro

ESTIMATIVA DE VITÓRIA EM DOTA 2 UTILIZANDO REDES NEURAIS ARTIFICIAIS

Monografia apresentada à Coordenação de Engenharia de Computação do Campus Timóteo do Centro Federal de Educação Tecnológica de Minas Gerais para obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: Douglas Nunes de Oliveira

Timóteo

2018

.

.

.

Dedico à aprendizagem.

Agradecimentos

À minha primeira orientadora, Márcia, pelo incentivo, dedicação e paciência.

Ao meu orientador, Douglas, pela confiança e autonomia.

Às boas amizades, que enobrecem e alegram a jornada.

À Rebeca, pelas boas memórias construídas.

“Ter todas as respostas significa apenas que você vem fazendo perguntas desinteressantes.”
Joey Comeau

Resumo

Este trabalho propõe-se a estimar qual time será vitorioso em uma partida de DOTA 2, baseando-se apenas nos personagens escolhidos. Um sistema capaz de fazer essas previsões pode ser utilizado como base para fazer recomendações aos jogadores quanto aos melhores heróis a se escolher em determinada situação. Para alcançar esse objetivo, foi realizado um levantamento de trabalhos existentes com propostas similares e, posteriormente, elaborado um modelo de rede neural artificial MLP que pudesse aprender a fazer previsões de vitória utilizando registros de partidas passadas. Para treinar o modelo, foram adquiridos 1,2 bilhão de registros de partidas e, a partir do refinamento destes, construiu-se uma base de dados com 800.000 amostras. Realizando o treinamento a partir desta base, o modelo atingiu 61,7% precisão em estimar o time vencedor. Utilizando, porém, uma parcela menor da base, foi possível alcançar uma maior precisão, chegando a 66,3%. Os resultados encontrados indicaram que as características da base de dados utilizada podem influenciar a taxa de precisão que se pode atingir e que, possivelmente, o modelo alcançaria um melhor desempenho caso treinasse numa base de dados ainda mais refinada.

Abstract

The current study attempts to design a model to predict the outcome of a DOTA 2 match, basing itself only on the heroes present on each team. Such a model could be used as a foundation for a hero recommendation engine that would help players in picking the most advantageous hero for a given scenario. In order to achieve this goal, we first surveyd the preexisting tools with similar purposes, and then designed a MLP neural network which could learn to make predictions based on historical match data. The dataset utilized had 800.000 samples, and was crafted by filtering and refining 1,2 billion records of matches. After training the model, it was able to correctly predict the outcome of 61,7% of the samples in the dataset. However we found that when utilizing a smaller share of the dataset, the network was able to improve its predictions' accuracy, peaking at 66,3%. Our findings suggest that not only some characteristics of the dataset used for inference could have a signifcant impact on the model's ability to learn and predict outcomes, but also that our model could yield better results given a more refined dataset to train uppon.

Lista de ilustrações

Lista de tabelas

Lista de quadros

Lista de abreviaturas e siglas

AMD	Advanced Micro Devices
AOT	Ahead-Of-Time
API	Application Program Interface
CPU	Central Processing Unit
DOTA	Defense of the Ancients
GB	Gigabyte
GPU	Graphic Processing Unit
JIT	Just-in-Time
JSON	JavaScript Object Notation
LTU	Linear Threshhold Unit
Máx.	Máximo
MLP	Multilayer Perceptron
MOBA	Multiplayer Online Battle Arena
RAM	Random Access Memory
RNA	Redes Neurais Artificiais
SGD	Stochastic Gradient Descent
XML	Extensible Markup Language

Sumário

1 Introdução

“A resposta certa, não importa nada:
o essencial é que as perguntas estejam certas”.
Mário Quintana

1.1 Contextualização

A indústria de jogos tem crescido nos últimos anos, movimentando 108 bilhões de dólares em 2017, e projeta-se que atingirá 128 bilhões em 2020 (SUPERDATA, 2017). Dentro desse universo, o gênero *Multiplayer Online Battle Arena* de jogos tem se tornado popular, com um total mensal aproximado de 140 milhões de usuários ativos em junho de 2015. DOTA 2 é, atualmente, o jogo do gênero supracitado com a segunda maior quantidade de jogadores (SUPERDATA, 2015).

0 maior torneio anual de DOTA 2 é chamado de *The International*. O valor total do prêmio neste torneio, em 2018, foi de 25,5 milhões de dólares: 11,1 milhões de dólares para a equipe vencedora, 4 milhões de dólares para a segunda colocada e o restante distribuído gradualmente até a décima oitava equipe (VALVE, 2018a).

Nesse jogo, duas equipes adversárias, cada qual com 5 jogadores humanos, enfrentam-se em uma partida, em que cada jogador controla um único personagem, denominado herói (GODEC, 2015). No começo da partida, o jogador seleciona o personagem que utilizará baseando-se em considerações como:

1. Experiência / aptidão do jogador com determinados personagens;
2. Sinergia com personagens de outros jogadores da própria equipe;
3. Personagens que tenham vantagem sobre os personagens da outra equipe;
4. Estratégia da equipe.

Embora não se possa afirmar que determinado herói seja superior a outro, cada um se adequa melhor a determinada situação e, com isso, pode obter vantagens ou desvantagens sobre os demais. Desse modo, um preciso preditor de vitória poderia ser utilizado como base para uma ferramenta relevante que auxilie o jogador numa decisão mais vantajosa de herói a se escolher (KINKADE, 2015).

A proposta deste projeto consiste, por conseguinte, na utilização de técnicas de inteligência artificial para estimar o time vencedor em uma partida ordinária de DOTA 2 pela escolha dos personagens.

1.2 Problema

Atualmente, embora existam 116 heróis diferentes disponíveis para escolha no jogo (VALVE, 2018c), não se pode dizer que exista algum que um seja melhor que outro. Em vez disso, o que se perceber na prática, é que cada um é se faz mais adequado em determinado cenário ou situação (GODEC, 2015).

Por exemplo, um herói pode ser melhor para uma estratégia que vise ganhar o jogo em curto tempo, mas outro pode se tornar muito relevante em jogos de duração mais longa. Alguns podem ter ênfase em dar suporte e ajuda a seus aliados, enquanto outros possuem como capacidade principal infligir muito dano em seus inimigos (GODEC, 2015).

Há ainda interações entre as habilidades dos personagens, de maneira que um pode ter uma vantagem inerente contra outro personagem específico. Cria-se, então, uma situação similar a um jogo de "Pedra, Papel & Tesoura", porém com interações abstratas e maior nível de complexidade.

A comunidade de jogadores reconhece que a escolha dos heróis é um fator de grande influência para o decorrer de uma partida (GODEC, 2015). Diante disso, pretende-se desenvolver um modelo que consiga extrair informações sobre os heróis e permitir que, para certa combinação de heróis escolhida em uma partida, se estime qual o time em vantagem e qual sua chance de vitória.

1.3 Objetivos

Propor um modelo capaz de estimar o time vencedor em uma partida de DOTA 2, com base nos personagens escolhidos pelas duas equipes no começo da partida. Um modelo com estas características pode ser utilizado como base para um sistema de recomendação de personagens ao time auxiliado, com o objetivo de obter vantagem técnica ou até mesmo para utilização em bancos de apostas.

1.4 Estrutura da monografia

Este trabalho está estruturado em 6 capítulos. O presente capítulo trata da introdução ao problema e propõe soluções, ao passo que o capítulo 2 apresenta as bases teóricas, o que inclui conceitos de funcionamento do jogo e fundamentação das técnica de inteligência artificial utilizadas.

O capítulo 3, por sua vez, apresenta o desenvolvimento e os procedimentos metodológicos, enquanto o capítulo 5 expõe os resultados obtidos ao longo do percurso. Finalmente, o capítulo 5 levanta interpretações que podem ser realizadas sobre os resultados e o capítulo 6 agrega as considerações finais.

2 Conceitos gerais

“O período de maior ganho em conhecimento e experiência
é o período mais difícil da vida”.
Dalai Lama

O objetivo principal deste capítulo é apresentar os fundamentos teóricos e metodológicos sobre os quais esta pesquisa é executada, além de mapear os recentes trabalhos desenvolvidas relacionadas a previsão de vitória, assim como as ferramentas existentes para auxílio de seleção de heróis.

2.1 Fundamentação teórica

Nas seções seguintes, serão apresentados os conceitos que fundamentam este trabalho. Na seção 2.1.1 é apresentado o jogo em questão, no qual se deseja estimar a vitória utilizando inteligência artificial, enquanto na seção 2.1.2 são explicados os conceitos de redes neurais e modelos clássicos. Por fim, as seções 2.1.3 a 2.1.9 explicam as tecnologias empregadas no desenvolvimento do presente trabalho.

2.1.1 DOTA 2

DOTA 2 é um jogo eletrônico do gênero *Action Real-Time Strategy*, ou também considerado do gênero *Multiplayer Online Battle Arena* (abreviado como *MOBA*), que foi desenvolvido pela Valve Corporation como sequência do *Defense of the Ancients* (*DOTA*). Este, por sua vez, é um jogo advindo da modificação em um mapa desenvolvido para *Warcraft III* que atingiu enorme popularidade (LEAHY, 2012).

O jogo supracitado é beaseado em partidas online, em que o objetivo primário é destruir a edificação que se localiza no centro da base inimiga e, com isso, derrotar o time adversário. No entanto, cada base é defendida por torres e ondas de unidades (chamadas *creeps*) que percorrem os caminhos principais do mapa (chamados *lanes*) (DEAN, 2011).

No começo de cada partida, o jogador escolhe para controlar um dos 116 personagens disponíveis no jogo (VALVE, 2018c). Cada um dos dois times é formado por 5 personagens, o que totaliza 10 integrantes da partida, que irão combater entre si ao longo do jogo (GODEC, 2015).

Os personagens são chamados heróis e apresentam habilidades e estilos de jogo únicos. No início de uma partida, todos os heróis iniciam no nível 1, podendo chegar até o nível 25 à medida que acumulam pontos de experiência. A cada nível avançado, os heróis recebem um ponto para aumentar seus atributos básicos ou melhorar suas habilidades especiais. Então, quanto maior o nível do herói, maiores são seus atributos e sua capacidade de impactar o jogo com suas habilidades (YIN-POOLE, 2011).

Ao longo da partida, o herói vai adquirindo ouro. Este corresponde à moeda do jogo e pode ser utilizado para comprar itens de aprimoramento – como aumento do dano, da vida ou da resistência à mágica – e até mesmo para reviver instantaneamente após a morte, uma vez que em circunstâncias normais, quando um herói morre, deve-se esperar 5 + 1,8x (sendo que x é o nível do respectivo herói) segundos para renascer.

Os heróis recebem passivamente 1,67 ouro por segundo e são recompensados com ouro adicional sempre que desferem o último golpe em qualquer unidade não-aliada do jogo, levando-a à morte. Não obstante, heróis inimigos geralmente fornecem mais ouro que outras unidades (GODEC, 2015).

Existem duas classes de habilidades no jogo: passivas e ativas. A maior parte das habilidades passivas permanecem em funcionamento contínuo, enquanto as ativas requerem que o jogador utilize-as manualmente. Em geral, as habilidades ativas requerem "mana", um recurso de todo herói que é utilizado para limitar o uso de habilidades e adicionar complexidade ao jogo, fazendo com que o jogador tenha que se atentar a um artifício adicional para controlar seu herói de maneira eficaz. A maioria das habilidades requer um tempo de recarga chamado de *cooldown*, isto é, o jogador que utilizar a habilidade deve esperar um certo tempo para utilizá-la novamente. Este tempo pode variar de 1,5 segundo a 260 segundos (GODEC, 2015).

As características intrínsecas de cada herói tornam-o mais conveniente para executar determinado papel na equipe e no jogo. Alguns dispõem de habilidades capazes de imobilizar os inimigos, impedindo que se movimentem, ataquem ou utilizem habilidades durante um curto período de tempo (geralmente de 1 a 3 segundos). Durante o jogo, estes 3 segundos podem ser suficientes para que heróis aliados encurralem e matem um inimigo, sem que ele tenha chance de escapar. Assim, heróis com este tipo de habilidade são chamados de *disablers* (debilitadores), por serem capazes de debilitar heróis oponentes (GODEC, 2015).

Quando heróis possuem habilidades debilitantes com *cooldown* relativamente baixo (menor que 30 segundos), eles podem ser considerados *gankers*, isto é, bons em praticar *ganks*. *Ganking* é o ato de se movimentar pelo mapa com o intuito de matar um herói inimigo incauto, utilizando-se da vantagem numérica (exemplo: dois heróis aliados contra um inimigo). Além disso, é utilizado para ganhar benefício em ouro e pontos de experiência (KINKADE, 2015).

2.1.2 Aprendizado de máquina

Quando a maioria das pessoas ouve os termos aprendizado de máquina, geralmente imaginam um robô com características humanoides, frequentemente representados em filmes e obras e ficção científica. Entretanto, aprendizado de máquina não é uma fantasia futurista, e sim algo atual, e que inclusive já existe há décadas em certas aplicações específicas como reconhecimento ótico de caracteres (GÉRON, 2017, p. 16).

De maneira geral, pode se definir aprendizado de máquina como "*(...) o campo de estudo que dá a computadores a habilidade de aprender sem precisarem ser explicitamente programados.*"[1] (SAMUEL, 1959, Tradução nossa).

Enquanto sob a ótica de engenharia, pode-se dizer que um programa aprende a partir de uma experiência ou exemplo, se estes contribuírem diretamente para uma melhora do desempenho deste programa em determinada tarefa (MITCHELL et al., 1997).

Por exemplo, os filtros de *email anti-spam* atuais, são programas de aprendizado de máquina que conseguem aprender a distinguir *emails* válidos de *emails spam*, desde que se forneçam a eles exemplos de *emails*, válidos e exemplos de *emails spam*. O conjunto de exemplos que os programas utilizam para treinar são denominados **base de teste**,

[1] [Machine Learning is the] field of study that gives computers the ability to learn without being explicitly programmed.

enquanto individualmente, cada exemplo também pode ser chamado de amostra (GÉRON, 2017, p. 17).

2.1.3 Redes neurais artificiais

Os conceitos modernos de redes neurais são datados do começo dos anos 40, quando foi possível provar que redes compostas por neurônios artificiais poderiam realizar, a princípio, a computação de qualquer problema lógico ou aritmético. Já na década de 50, a primeira rede *Perceptron* foi desenvolvida por Frank Rosenblatt (ROSENBLATT, 1960), demonstrando a capacidade de reconhecimento de padrões. Posteriormente, os métodos para a criação dessas estruturas foram aperfeiçoados e novas ideias foram apresentadas, como o conceito de *Backpropagation*, que é um método para mitigar erros durante treinamento supervisionado. Dessa forma, de acordo com a saída do algoritmo e um valor esperado para tal, é possível calcular o erro gerado pela rede, que então é propagado às camadas anteriores, corrigindo a saída dos neurônios para em iterações posteriores, se aproximarem melhor do resultado esperado (WERBOS, 1990).

- Neurônio - É o elemento básico de uma rede neural. Neurônios artificiais podem possuir dois ou mais receptores de entrada, em que cada um é encarregado de receber um sinal específico. Após o processamento do sinal de entrada, o mesmo retorna um valor binário de saída, representando 'sim' e 'não'. Como cada neurônio é capaz de receber *N* entradas, este tem a capacidade de 'perceber' diferentes características necessárias para a resolução de um determinado problema. Um neurônio pode ser condicionado a diferenciar objetos e terá diferentes sinais de entrada, um para cada característica do tipo de objeto a ser identificado – como, por exemplo, cor e formato – e, a partir das informações dadas na entrada, o mesmo gera um sinal de saída. A partir desse sinal, é possível realizar a identificação do objeto.
- Conexões - São ligações entre uma ou mais saídas oriundas de N neurônios, direcionadas à entrada de um nó pertencente a outra camada da rede. Essas conexões possuem um peso, diretamente ligado à força de conexão entre os nós. A magnitude de uma conexão é responsável também por determinar o grau de influência de um nó em relação aos demais ligados a ele. Entradas cuja as conexões possuem peso aproximado de zero não alteram os valores de saída. Valores negativos nas conexões significam que, aumentando o grau de entrada, os valores de saída diminuem. As conexões também são responsáveis por propagarem, através das camadas da rede, as informações referentes às saídas esperadas para determinada entrada. De certa forma, é possível afirmar que os pesos das arestas são responsáveis por armazenar o conhecimento representado no modelo. Dessa forma, a estrutura geral de uma rede neural representa a estrutura física de um cérebro (BRAGA; CARVALHO; LUDERMIR, 2000).

Uma rede neural artificial é um sistema paralelo distribuído composto por unidades de processamento simples. Estas unidades são dispostas em uma ou mais camadas e são interligadas por um grande número de conexões, geralmente associadas a pesos. Os pesos armazenam o conhecimento representado no modelo e servem para ponderar a entrada recebida por cada neurônio da rede. O funcionamento destas redes é inspirado em uma estrutura física concebida pela natureza: o cérebro humano (BRAGA; CARVALHO; LUDERMIR, 2000).

2.1.4 Perceptron

O Perceptron é um modelo de rede neural artificial clássico em aprendizagem de máquina. De outro modo, é um algoritmo de aprendizagem supervisionada que trabalha com classificação, sendo capaz de decidir se um conjunto de dados de entrada pertence a uma determinada classe (FREUND; SCHAPIRE, 1999). O Perceptron é baseado em uma abordagem diferente da proposta por um neurônio artificial. O mesmo utiliza uma estrutura chamada de *Linear Threshhold Unit* (*LTU*). A partir desse conceito, a entrada e saída passam a ser valores numéricos, ao invés de binários *On/Off* (GÉRON, 2017, p. 322). Dessa forma, cada entrada é associada a um peso e a *LTU* realiza o somatório dos valores de entrada e aplica uma função denominada *Step Function* ao valor resultante. A Figura 1 demonstra abordagens comumente utilizadas em Perceptrons.

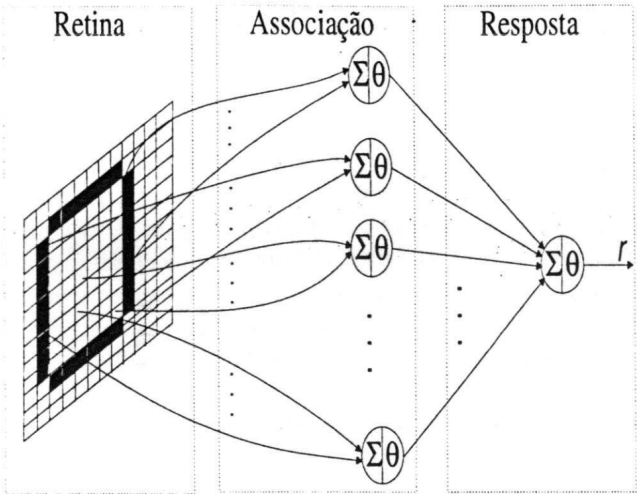
Figura 1 – *Step Functions* comuns em Perceptrons

$$\text{heaviside}\,(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \qquad \text{sgn}\,(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

Fonte – (GÉRON, 2017, p. 322).

Uma única *LTU* pode ser usada para realizar uma simples classificação linear binária. A mesma calcula, através de uma combinação de entradas lineares, um resultado. Caso o mesmo exceda o valor máximo, é retornada uma classe positiva na saída; caso contrário, negativa (GÉRON, 2017, p. 322). O processo ao qual a *LTU* é condicionada a realizar tal tarefa é denominado aprendizagem supervisionada. Baseando-se nessas informações, o algoritmo treina a rede de maneira supervisionada, verificando a diferença entre a saída da rede e uma saída conhecida. O objetivo é ajustar os parâmetros da rede, de forma a encontrar uma ligação entre os dados de entrada e saída fornecidos (BRAGA; CARVALHO; LUDERMIR, 2000). A Figura 2 ilustra este modelo de funcionamento.

Figura 2 – Topologia de um Perceptron simples com única saída



Fonte – (BRAGA; CARVALHO; LUDERMIR, 2000, p. 35).

Considerando um nodo qualquer da camada de resposta de um perceptron e seus vetores de entrada x’ e de pesos w’, sua ativação é dada pela Equação 2.1 aplicada a uma

porta limiar, sendo geralmente a função sigmoide a mais utilizada. De uma maneira simples, o processo de aprendizado consiste em obter um valor deltaW que, somado ao vetor de pesos w', torne o resultado da ativação mais próximo do resultado desejado.

$$\sum_{i=1}^{m} w_i' x_i' = 1 \tag{2.1}$$

2.1.5 Perceptron de múltiplas camadas

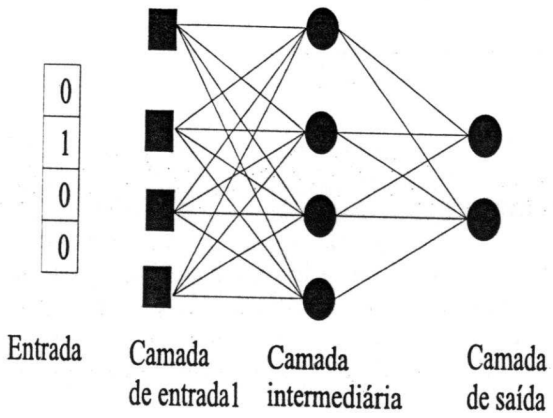
Um Perceptron de múltiplas camadas é um modelo clássico de rede neural, que utiliza uma ou mais camadas intermediárias. Ou seja, é uma modificação do modelo do Perceptron padrão, que consegue distinguir dados que não são linearmente separáveis. Segundo (CYBENKO, 1989), uma rede com uma camada intermediária pode implementar qualquer função contínua, enquanto a utilização de duas permite a aproximação de qualquer função matemática.

> "Pode ser dito que as unidades intermediárias de uma rede MLP funcionam como detectores de características. Elas geram uma codificação interna dos padrões de entrada, que é então utilizada para a definição da rede. Dado um número suficientemente grande de unidades intermediárias, é possível formar representações internas para qualquer conjunto de padrões de entrada."(BRAGA; CARVALHO; LUDERMIR, 2000).

Todavia, a utilização de muitas camadas intermediárias pode ser problemática, visto que, quanto maior a distância de uma camada até a camada de saída, menos útil se torna o erro medido durante o treinamento (BRAGA; CARVALHO; LUDERMIR, 2000).

Uma rede MLP consiste em múltiplas camadas de nós em um grafo dirigido, com cada camada completamente conectada à próxima. Com exceção dos nós de entrada, cada nó é um neurônio, com uma função de ativação não-linear (BRAGA; CARVALHO; LUDERMIR, 2000). A Figura 3 ilustra uma rede MLP típica.

Figura 3 – Rede MLP típica com uma camada intermediária.



Fonte – (BRAGA; CARVALHO; LUDERMIR, 2000, p. 52).

O treinamento de uma rede MLP geralmente utiliza um algoritmo capaz de aplicar uma retro-propagação do erro identificado na camada de saída, chamado de *backpropagation*. De maneira simples, este algoritmo pode ser descrito no seguinte pseudocódigo:

```
INICIO
Inicializar pesos aleatoriamente;
ENQUANTO quantidade limite de iteracoes nao for atingido:
   PARA cada camada ate a final:
      Multiplicar o vetor de entrada por vetor de pesos de cada neuronio;
      Aplicar a funcao de ativacao no resultado;
      Propagar a saida de cada neuronio para os neuronios da proxima
         camada;
   Comparar a saida da camada final da rede com a saida esperada;
   Calcular o erro cometido pela rede;
   Atualizar os pesos da camada de saida com base no erro;
   PARA cada camada ate a de entrada:
      Retropropagar o erro para a camada anterior;
      Atualizar os pesos dos neuronios;
FIM
```

2.1.6 Funções otimizadoras

O processo de aprendizagem em si de uma rede neural artificial é regido majoritariamente pela função otimizadora, que é responsável por minimizar o erro de saída através de ajustes nos pesos. Geralmente a função objetivo que se deseja minimizar é o erro quadrático médio entre a saída esperada e a obtida. Abaixo descrevemos as funções otimizadores mais relevantes na literatura moderna.

2.1.6.1 Gradient Descent

O algoritmo *Gradient Descent* é um algorítimo genérico de otimização capaz de encontrar soluções ótimas para uma ampla gama de problemas. A ideia central do GD é ajustar gradualmente os parâmetros de uma função objetivo, na direção mais íngreme do gradiente – isto é, onde o a derivada parcial da função apresentar maior módulo (GÉRON, 2017, p. 150).

Um importante parâmetro do *Gradiant Descent* é a taxa de aprendizado. Se esta for muito baixa, o algoritmo levaria muito tempo para convergir. Em contrapartida, se for muito alta, é possível que o algoritmo ultrapasse o ponto mínimo e acabe chegando em um ponto ainda mais alto do que onde começou (GÉRON, 2017, p.152).

2.1.6.2 SGD

Um problema com a implementação do *Gradiant Descent* é a necessidade de se computar a derivada parcial da função objetivo em relação a cada parâmetro do modelo, levando em consideração todas as amostras da base de treino em cada iteração. Isso faz com que o algoritmo apresente um desempenho muito lento em situações que a base de treino é grande. No extremo oposto, a variação desse algoritmo chamada *Stochastic Gradient Descent* escolhe uma única amostra aleatória da base de treino e calcula os gradientes baseando-se apenas naquela amostra (GÉRON, 2017, p. 157).

Isso torna possível realizar treinos utilizando bases de treino maiores, já que apenas uma amostra precisa estar na memória por iteração, e diminui o tempo gasto por iteração, uma vez que passa a manipular uma quantidade muito menor de dados.

Por outro lado, a natureza estocástica do algoritmo torna ele bem menos regular que o GD tradicional: ao invés de decrescer a função objetivo suavemente até atingir o mínimo, os valores oscilam constantemente, decrescendo apenas em média. Com o passar do tempo, ela converge na direção do mínimo, mas nunca o atinge precisamente (GÉRON, 2017, p. 158).

Uma segunda variação do algoritmo *Gradient Descent* é o *Mini-batch Stochastic Gradient Descent*, que é relativamente simples de compreender quando se conhece os anteriores. Em cada iteração, ao invés de utilizar uma única amostra aleatória ou a base de treino integralmente, utiliza-se *batches* de subconjuntos aleatórios da base de treino.

Dessa maneira, o algoritmo *Mini-batch Stochastic Gradient Descent* é capaz de se sobrepor às limitações dos anteriores. A sigla SGD é utilizada neste trabalho para se referir ao *Mini-batch Stochastic Gradient Descent*, pois esta é a implementação genérica do algoritmo.

Caso o tamanho do *batch* escolhido seja 1, ele se comporta como o *Stochastic Gradient Descent*. Caso seja igual ao tamanho da base de treino, se comporta como o *Gradiant Descent*. Para qualquer outro tamanho entre esses dois, se comporta como um *Mini-batch Stochastic Gradient Descent* (GÉRON, 2017, p. 161).

2.1.6.3 AdaGrad

O algoritmo Adagrad também é similar ao SGD e pode ser entendido como uma variante que fornece ao algoritmo a capacidade de memorizar todo o percurso de seu progresso. Dessa forma, permite variar as taxas de aprendizado de maneira diferente para cada dimensão da função, reduzindo a taxa de aprendizado para dimensões que tiveram gradientes maiores ao longo do treinamento.

Na prática, isso ajuda a direcionar as atualizações de pesos de forma mais direta para o ponto em que a função atingirá o mínimo global, mas pode acabar reduzindo excessivamente a taxa de aprendizado. Sendo assim, em alguns casos, pode fazer com que a velocidade de convergência se torne proibitivamente lenta (GÉRON, 2017, p. 373).

2.1.6.4 RMSProp

O RMSProp, por sua vez, tem seu funcionamento de forma quase idêntica ao AdaGrad, porém tenta resolver o problema da redução excessiva da taxa de aprendizado, de forma a limitar a memória do algoritmo e utilizar apenas suas iterações mais recentes. Desse modo, evita a redução cumulativa da taxa de aprendizado no decorrer do treinamento (GÉRON, 2017, p. 374).

2.1.6.5 Adam

Assim como o RMSProp é uma variação do AdaGrad, o otimizador Adam pode ser visto como uma variação do RMSProp, diferenciando-se pela inclusão de uma outra técnica chamada *Momentum* – que faz analogia com o termo 'momento', utilizado em Física.

A ideia central consiste em adicionar um multiplicador variável para a taxa de aprendizado. O algoritmo, então, identifica quando a direção do gradiente escolhida pelo otimizador está afetando positivamente o resultado e segue aumentando gradativamente a taxa de aprendizado, enquanto a direção escolhida estiver contribuindo positivamente para a otimização. Por outro lado, quando o algoritmo detecta que está caminhando em uma direção ruim, ele reduz a taxa de aprendizado (GÉRON, 2017, p. 369).

2.1.6.6 Nadam

Finalmente, o otimizador Nadam tem princípio de funcionamento exatamente igual ao Adam, porém utiliza uma variação especial no momento do cálculo do *Momentum*, proposta por (NESTEROV, 1983). A variação consiste em medir o gradiente da função objetivo na direção do *Momentum* anterior, em uma posição ligeiramente à frente da atual na qual se encontra.

2.1.7 Funções de ativação dos neurônios

As funções de ativação podem ter um impacto expressivo no desempenho de uma rede neural artificial. Nesta seção, será feira uma breve revisão sobre as alternativas mais relevantes e, em seguida, será apresentada a topologia inicial adotada.

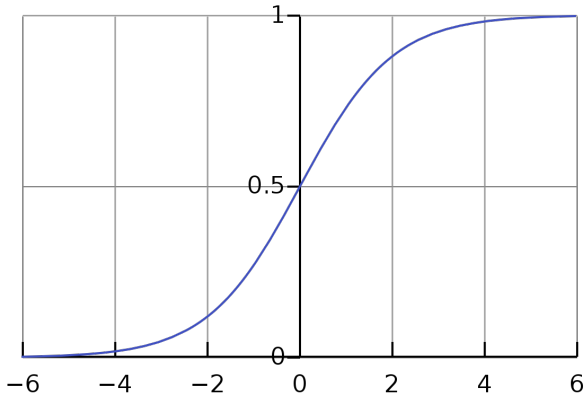
2.1.7.1 Sigmoidal

A função sigmoidal – também chamada de função logística – é uma função de ativação que faz uma boa representação do processo de ativação que ocorre em neurônios biológicos. A função pode ser definida pela Equação 2.2 e ilustrada pela Figura 4.

$$y = \frac{e^x}{1 + e^x} \tag{2.2}$$

Por muitos anos seguintes à concepção das redes MLP, a função de ativação sigmoidal permaneceu sendo a mais utilizada e recomendada para treinamento acerca de RNA(s) na literatura (BRAGA; CARVALHO; LUDERMIR, 2000, p. 51) (GLOROT; BORDES; BENGIO, 2011, p. 316). Entretanto, na última década, houve um grande declive na utilização, dando lugar a outras funções que consistentemente demonstraram apresentar resultados melhores.

Figura 4 – Gráfico genérico da função sigmoide



Fonte – (SCIENCE, 2018).

Por exemplo, até 2006, o treinamento bem sucedido de redes MLP com muitas camadas era virtualmente inédito na comunidade acadêmica, mas inúmeros algoritmos capazes de treiná-las foram desenvolvidos desde então. Nesse mesmo processo, descobriu-se que a função sigmoidal não é apropriada para uso nas camadas intermediárias de redes com muitas camadas, pois ela tende a causar a saturação da primeira camada intermediária (GLOROT; BENGIO, 2010). Apesar disso, ela continua sendo útil como função de ativação da camada de saída quando só existem duas classes de saída (GÉRON, 2017, p. 276).

2.1.7.2 Tangente hiperbólica

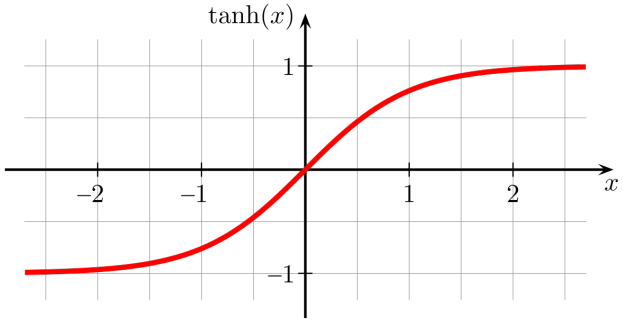
A função tangente hiperbólica é uma função monótona diferenciável muito similar à sigmoidal, podendo inclusive ser compreendida como a função sigmoidal representada em outra escala, de tal forma que pode ser interpretada por ambas as Equações 2.3 e 2.4.

$$tanh(x) = 2 * sigmoidal(2x) - 1 \tag{2.3}$$

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.4}$$

Apesar da similaridade, a função tangente hiperbólica traz muitas vantagens em relação à sigmoidal. Sua imagem é [-1,1], seu eixo X cruza com o Y no ponto zero e ela é antissimétrica, como representado pelo gráfico da Figura 5. As entradas negativas terão saídas negativas, bem como entradas próximas de zero serão mapeadas próximas de zero na saída. Essas características são benéficas do ponto de vista do processo de otimização (GLOROT; BORDES; BENGIO, 2011, p. 317) (O'CONNOR, 2013) (SHARMA, 2017) (OGUNMOLU, 2017).

Figura 5 – Gráfico genérico da função tangente hiperbólica



Fonte – (WIKIPEDIA, 2018b).

2.1.7.3 ReLU *(Rectified Linear Unit)*

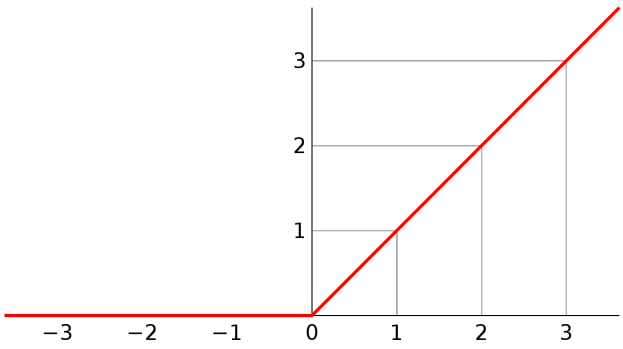
A função de ativação ReLU – ilustrada na Figura 6 – é uma das mais utilizadas no mundo atualmente, em vista dos bons resultados que apresenta em *Deep Learning* e redes neurais convolucionais. A função em si é simples e pode ser descrita como uma função de máximo: $y = max(0, x)$. É contínua, porém não diferenciável em $x = 0$. Em teoria, isto é algo ruim para o método de funcionamento do *backpropagation*, já que este depende da derivada. Por este motivo, durante muito tempo o uso da função ReLU não foi cogitado. Entretanto, na prática, ela funciona muito bem e possui a vantagem de ser mais rápida para se computar (GÉRON, 2017, p. 330).

A função ReLU possui algumas outras propriedades que tornam-a atrativa para o uso em RNA(s). Destas, destacam-se:

- Funções como a tangente hiperbólica raramente "desativam", isto é, fornecem saída nula. Porém, já foi demonstrado que biologicamente e em redes profundas, é necessária a desativação de neurônios, também chamada de ativação esparsa (GLOROT; BORDES; BENGIO, 2011, p. 316);

- O fato da função não ter um valor máximo de saída ajuda a reduzir alguns problemas durante a otimização pelo método do gradiente (GÉRON, 2017, p. 328);
- Sua derivada é constante, o que elimina o problema de saturação dos gradientes (*Vanishing Gradient*). Por exemplo, durante o *backpropagation*, caso seja passada uma saída muito alta para as funções sigmoidal ou tangente hiperbólica, a derivada destas será zero, impedindo o aprendizado da rede nas camadas anteriores (GLOROT; BORDES; BENGIO, 2011, p. 318).

Figura 6 – Gráfico genérico da função ReLU

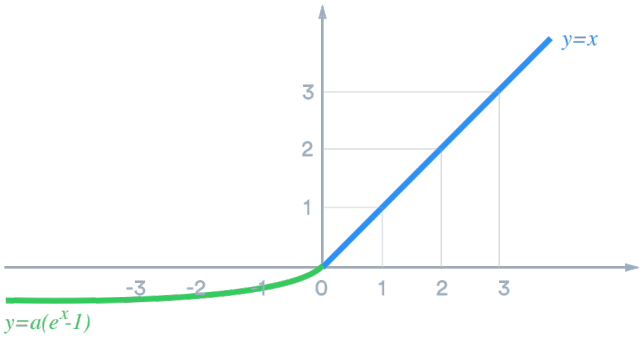


Fonte – (WIKIPEDIA, 2018a).

2.1.7.4 Leaky ReLU

Apesar de ter muitas vantagens, a função ReLU não é perfeita. A mesma característica que permite a ela apresentar a ativação esparsa, também pode levar a um problema de excessiva morte de neurônios – uma situação onde certa quantidade de neurônios da rede é permanentemente desligada durante o treino, pois passa a apresentar somente zero como saída. Para aliviar esta condição, pode-se utilizar uma variante da função ReLU, como a LeakyReLU, que está representada pela Figura 7 e pode ser definida como $LeakyReLUa(z) = max(az, z)$ (GÉRON, 2017, p. 348).

Figura 7 – Gráfico genérico da função LeakyReLU



Fonte – (MIND, 2018).

(XU et al., 2015) investigaram a performance de diferentes variantes da função ReLU como funções ativadoras em uma RNA convolucional para um problema tradicional de classificação de imagens. Seus resultados indicaram que houve uma melhora consistente nos resultados ao utilizar uma variante da função ReLU com gradiente não nulo para a parte

negativa do domínio. Contudo, é relevante ressaltar que (GLOROT; BORDES; BENGIO, 2011, p. 321) sugere que funções de ativação retificadoras são mais adequadas para problemas relacionados a imagens.

2.1.8 *Overfitting*

Embora redes neurais profundas com muitos parâmetros de entrada possuam uma enorme capacidade de aprendizado, *overfitting* pode ser um sério problema para estas redes. *Overfitting* é o termo utilizado para descrever uma rede que apresenta alto grau de acerto para as amostras com as quais ela está sendo treinada, porém fornece resultados ruins ao tentar fazer uma previsão sobre uma amostra a qual ela nunca tenha sido exposta antes. A lógica por trás deste comportamento pode ser entendida de uma maneira simples: é como se a rede tivesse decorado o resultado esperado para todas as amostras treinadas, ao invés de realmente aprender a resolver o problema de maneira genérica (SRIVASTAVA et al., 2014) (GÉRON, 2017, p. 44) (MATHWORKS, 2018).

Overfitting geralmente acontece em situações em que o modelo é muito complexo se comparado à quantidade de amostras disponíveis para treino ou de ruído presente nos dados de treinamento. As soluções para evitar o *overfitting* baseiam-se nas seguintes ideias principais:

- Simplificar ou limitar o modelo;
- Reduzir o ruído nas amostras de treino;
- Obter mais amostras para treino.

Dessa forma, quanto maior a rede utilizada, maior sua capacidade de criar funções complexas. Utilizando uma rede pequena o suficiente, ela não terá capacidade suficiente para atingir *overfitting* (MATHWORKS, 2018).

2.1.9 Parada antecipada

Uma das maneiras mais simples de contornar os efeitos de overfitting é a técnica do *Early Stopping*, ou parada antecipada. Esta técnica necessariamente requer que a base de dados em estudo seja dividida em um grupo de amostras que serão utilizadas para treino e outro de amostras a serem utilizadas somente para teste durante o treino. As amostras neste último grupo não irão contribuir com o aprendizado da rede em si, mas apenas para investigar o quão boas estão sendo as previsões da rede quando são fornecidas amostras que ela nunca encontrou antes.

Monitorando a taxa de acerto da rede para os dois grupos supracitados, é possível identificar quando a rede está começando a fazer *overfit* e interromper o treinamento. Durante o treino, a taxa de acerto nos dois grupos usualmente sobe, enquanto a taxa de erro médio cai. Entretanto, quando começa a ocorrer *overfit*, a taxa de acerto no grupo de teste tipicamente começa a cair, visto que a rede está deixando de generalizar e passando a apenas decorar as amostras. Quando a taxa de precisão no grupo de teste permanece em queda durante um número N de iterações ou não apresenta melhora, enquanto a taxa de precisão no grupo de treino continua a subir, o treinamento é interrompido e são restaurados os pesos do momento em que a rede apresentava o melhor resultado para a base de teste (MATHWORKS, 2018) (GÉRON, 2017, p. 177).

2.1.10 *Dropout*

Dropout é o nome dado a uma técnica desenvolvida em 2014 com o objetivo principal de lidar com o problema de *overfitting*. A ideia chave consiste em desativar neurônios aleatórios durante o treinamento da rede, impedindo-os de coadaptarem de forma excessiva. A técnica é capaz de reduzir significativamente o *overfitting* e apresentou resultados melhores que os outros métodos utilizados até então. Foi demonstrado que é capaz de melhorar o desempenho de redes neurais em tarefas de aprendizado supervisionado em visão, reconhecimento de fala, classificação de documentos e biologia computacional (SRIVASTAVA et al., 2014).

2.1.11 Keras

Trata-se de um *framework* especializado para implementação de redes neurais. O objetivo do Keras é oferecer simplicidade na codificação e reduzir ao máximo o tempo entre a concepção da ideia até a execução do código, acelerando a etapa de prototipação, que é muito importante no desenvolvimento de redes neurais (CHOLLET et al., 2015).

2.1.12 TensorFlow

O TensorFlow foi concebido como uma biblioteca de código aberto, desenvolvido originalmente por engenheiros e pesquisadores da divisão de inteligência artificial do Google e utilizado para aplicações de aprendizado de máquinas e redes neurais, por exemplo. Possui arquitetura flexível, o que permite a implantação em várias plataformas que utilizem CPUs e GPUs, como desktops, clusters de servidores, dispositivos móveis e unidades de processamento especializadas em redes neurais (ABADI et al., 2016).

2.1.13 PlaidML

O PlaidML é uma plataforma criada em 2017 que se propõe a tornar viável a utilização de *deep learning* em qualquer tipo de dispositivo (NG, 2017). Sua vantagem sobre outras plataformas populares é a capacidade de ser utilizado com qualquer GPU, independentemente de modelo ou fabricante. Difere-se do TensorFlow, portanto, que só fornece suporte a GPUs da NVIDIA, pois pode acelerar *deep learning* em GPUs AMD, Intel, NVIDIA, ARM ou ainda GPUs embarcadas. Essa característica faz com que em muitos casos ele seja mais rápido que o TensorFlow, já que este muitas vezes fica restrito aos CPU's (LAUB BRIAN RETFORD et al., 2017).

2.1.14 MATLAB

O MATLAB é uma plataforma de programação desenvolvida especialmente para engenheiros e cientistas. O núcleo do MATLAB é a linguagem de mesmo nome. Com ênfase em matrizes, esta permite que se expresse matemática computacional de maneira relativamente bem natural.

A plataforma é utilizada por milhões de engenheiros e cientistas tanto na indústria quanto na academia, e pode ser utilizado em um amplo campo de aplicações, como aprendizado de máquina, processamento de sinais, processamento de imagem e vídeo, sistemas de controle, biologia computacional entre muitas outras (MATHWORKS, 2017).

2.1.15 Python

O Python é uma linguagem de programação de alto nível orientada a objetos (PYTHON, 2018), lançada em 1991 por Guido van Rossum. Dentre as facilidades oferecidas pela linguagem, o uso de bibliotecas para a realização de tarefas específicas e a execução da aplicação sem a necessidade do passo de compilação são atrativos. Bibliotecas Python são coleções de funções que permitem a utilização de ações sem a necessidade da criação de trechos específicos de código para tal. *Data science* e *machine learning* são tópicos muito explorados nos últimos anos em tecnologia da informação. Bibliotecas existentes para esses tipos de análise comumente possuem uma interface Python, tornando-se uma das linguagens mais populares entre programadores (TECHLABS, 2016).

2.1.16 NumPy

O NumPy é o pacote fundamental de computação científica para Python (OLIPHANT, 2006), oferecendo diversos recursos como:

- Um objeto *array* para N dimensões versátil e eficiente;
- Ferramentas integradas para operações de álgebra linear e transformada de *Fourier*;
- Funcionalidades para geração e uso de números aleatórios.

Além dos usos científicos, NumPy também pode ser empregado como um eficiente contêiner multidimensional para dados genéricos de qualquer tipo. Isto permite a integração rápida e simples do NumPy com uma ampla variedade de base de dados (ASCHER et al., 2001).

2.1.17 Numba

O Numba é um compilador *Just-in-Time* de código aberto que traduz um subconjunto de códigos Python e NumPy em código de máquina. O Numba é projetado para ser usado com *arrays* e funções do NumPy, sendo que gera códigos para diferentes tipos de dados e layouts para otimizar o desempenho (LAM; PITROU; SEIBERT, 2015).

2.2 Trabalho relacionado

Foi realizada uma ampla pesquisa em busca de ferramentas que se propusessem a estimar a vitória ou recomendar heróis mais vantajosos a se escolher em uma partida de DOTA 2. Posteriormente, foram pesquisados os diferentes tipos de algoritmos de inteligência artificial e seus distintos fins e propriedades de aplicação.

Durante a pesquisa, pôde-se observar que, apesar de haver um interesse sobre o problema na comunidade de jogadores, há poucos estudos disponíveis, a ainda a maior parte destes são informais e não apresentam uma metodologia científica clara.

O único trabalho de cunho científico encontrado a respeito do tópico, foi o realizado por (CONLEY; PERRY, 2013). Nesta pesquisa, os autores afirmam ter obtido em torno de 69% de precisão em estimativa de vitória baseando-se apenas nos personagens selecionados por cada time, utilizando a técnica de regressão logística, otimizada pelo algoritmo *K-nearest neighbors*.

2.2.1 Ferramentas de recomendação de seleção

Apesar da pouca disponibilidade de artigos científicos sobro o tema, puderam ser encontradas diversas ferramentas que se propunham a auxiliar o jogador em sua decisão durante a fase de escolha dos heróis. As recomendações fornecidas por estas ferramentas são feitas de forma a tentar maximizar as vantagens sobre a equipe adversária e, consequentemente, aumentar as chances de vitória.

O princípio de funcionamento dessas ferramentas são de interesse para este trabalho, uma vez que também quantificam as características abstratas dos personagens e se baseiam na premissa de que um time pode ter uma maior chance de vitória sobre o outro pela escolha dos personagens. Dentre as ferramentas encontradas, destaca-se as seguintes:

2.2.2 Dota Edge

É um tipo de ferramenta que modela o problema a partir da interação de um par de personagens e da vantagem que um tem sobre o outro. Sua operação consiste nos seguintes passos:

- O usuário especifica os personagens já escolhidos pelo time opositor;
- O sistema avalia a vantagem relativa de cada personagem ainda disponível para escolha, em relação a cada personagem já escolhido pelo time opositor.

Essas vantagens relativas advém de *inputs* dos próprios usuários do sítio. Se eles optarem por contribuir com o sistema, podem escolher dois personagens e dizer qual tem vantagem sobre o outro, atribuindo uma nota arbitrária de 1 a 5, significando o quão boa é a suposta vantagem (DOTAEDGE, 2015).

2.2.3 True Picker

Esta ferramenta é baseada na interação de protocooperação entre dois ou mais personagens em um mesmo time, isto é, dois personagens que funcionam bem em conjunto. Por exemplo, um personagem que tenha muita facilidade em infligir dano físico aos inimigos trabalhando em conjunto com um personagem que tenha capacidade de ampliar o dano físico que os inimigos recebem. A ferramenta não leva em consideração a composição dos times inimigo e aliado. Método de operação:

- O usuário seleciona algum personagem e, então, a ferramenta irá sugerir possibilidades de outros personagens que têm uma boa relação de sinergia com o personagem selecionado;
- As supostas relações de sinergia de um personagem com outro são enviadas ao sistema por sugestão dos usuários do próprio sistema;
- Após recebida uma nova relação de sinergia entre dois personagens, outros usuários do sistema podem votar se gostaram daquela sugestão, com "Sim"ou "Não"(VIKTOROVICH, 2015).

2.2.4 Dota Picker

Esta ferramente também é fundamentalmente baseada na vantagem relativa da interação de um par de personagens. A operação acontece da seguinte forma:

- O usuário informa os personagens escolhidos no seu time e no time opositor;
- O sistema avalia a vantagem relativa e a sinergia de cada personagem ainda disponível em relação a cada personagem do time opositor, bem como o somatório das sinergias com os personagens do time aliado;
- São, então, sugeridos ao usuário os personagens que apresentaram a maior pontuação, isto é, cuja soma de vantagens e sinergia for maior.

As vantagens relativas advêm das análises de milhares de partidas, que são realizadas da seguinte maneira:

- Verifica-se a taxa média de vitória de um dado personagem em todos os jogos em que ele aparece;
- Verifica-se a taxa média de vitória deste mesmo personagem em todos os jogos em que um outro personagem X se encontra no time inimigo.

A partir da diferença das duas taxas, supõe-se uma vantagem relativa de um personagem sobre o outro.

As sinergias são definidas arbitrariamente pelos próprios desenvolvedores do sistema, baseados em seu conhecimento dos personagens.

2.2.5 DotaTeam

Esta ferramenta tem uma abordagem diferente das demais, pois se baseia no somatório das características do time, ao invés de analisar as interações entre os pares de heróis.

Os autores criaram uma série de áreas de especialidades e atribuíram a cada personagem uma nota, significando o quão bom ou ruim é naquela área de especialidade (exemplo: capacidade de debilitar um oponente durante confrontos entre times e capacidade de infligir danos às edificações do time inimigo). A partir das áreas de especialidades, os autores criaram uma heurística que almeja indicar quando a composição de um time está adequada.

A heurística leva em conta a soma das capacidades de um time em cada uma das áreas de especialidades, que deve estar dentro de um intervalo que os autores julgarem ser o ideal. A falta ou excesso de alguma área de especialidade significa, provavelmente, que a composição do time não está apropriada. A ferramenta também apresenta ao usuário uma relação entre a taxa de vitória dos personagens com o tempo de duração do jogo, indicando quando um personagem é melhor para jogos de curta ou longa duração. Porém, a ferramenta parece não contemplar estes dados para sugestão de personagens.

Apesar de trazer mecanismos interessantes, a ferramenta não leva em consideração a composição do time inimigo. Isto reduz sua funcionalidade prática, visto que, mesmo para um time excelente, pode existir uma combinação de heróis que o deixa em forte desvantagem (ELLIS, 2013).

2.2.6 Análise crítica das ferramentas

Para fins de auxílio na seleção de personagens em DOTA 2, existem atualmente ferramentas que trabalham com diferentes abordagens. Algumas têm como foco apenas interações específicas entre heróis aliados e heróis inimigos, sugeridas por usuários. Outras levam em consideração, além das supracitadas, as interações entre heróis já existentes no time aliado e os heróis ainda disponíveis para seleção. Um outro tipo leva em consideração somente a composição do time aliado, somando as características dos heróis que o compõem e verificando se o time possui personagens que possam cumprir adequadamente papéis considerados importantes.

Todas as ferramentas levam em consideração aspectos importantes para a escolha mais adequada de um personagem, porém elas focam individualmente em um aspecto e desconsideram outros fatores importantes. O resultado é uma saída não muito adequada, assim como saídas divergentes provindas de cada uma das ferramentes quando submetidas a uma entrada similar. Além disso, nenhuma das ferramentas encontradas demostra ter realizado testes de validação de suas saídas utilizando composições de times reais.

O Quadro 1 sumariza o método de funcionamento das ferramentas existentes. A primeira linha descreve alguns dos possíveis aspectos relevantes para escolha, enquanto a primeira coluna traz os nomes das ferramentas.

Quadro 1 – Quadro comparativo do método de funcionamento das ferramentes de auxílio de seleção de heróis

	Interações específicas "2-a-2" entre heróis amigo *vs* inimigo (*Input* manual)	Taxa de vitória de heróis "2-a-2" baseada em registros de partidas	Interações específicas entre heróis aliados "2-a-2"	Momento da partida em que cada herói é mais forte	Heurística de atributos genéricos de cada herói
Dota Edge	Sim	Não	Não	Não	Não
Truepicker	Sim	Não	Não	Não	Não
Dota Picker	Sim	Sim	Sim	Não	Não
DotaTeam	Não	Não	Não	Não	Sim

Fonte – Elaborada pelo autor.

3 Desenvolvimento e procedimentos metodológicos

”Nenhum esforço que vale a pena é simples em perspectiva;
se estiver correto, ele será simples em retrospectiva.”
Edward Teller

Abaixo, são sumarizados os procedimentos seguidos no decorrer da presente pesquisa, de forma geral e não exaustiva.

- Em contato com jogadores de DOTA 2, identificou-se o problema e foi idealizada uma proposta de objetivo que contribuísse para solucioná-lo;
- Realizou-se uma pesquisa, procurando levantar quais recursos e ferramentas existiam até o momento, que possuíssem o intuito de auxiliar na resolução do problema identificado;
- Procurou-se saber quais dados eram possíveis de se obter para serem utilizados num modelo que tratasse o problema;
- Percebida a possibilidade de se adquirir milhares de registros de partidas reais, buscou-se identificar qual técnica dentro do campo de aprendizagem de máquina se adequaria melhor ao problema;
- Notada a viabilidade do uso de redes neurais artificiais de aprendizagem supervisionada, pesquisou-se como poderiam ser codificadas as amostras de registros de partidas, de forma a serem utilizadas como entrada numa rede neural;
- Adquiriu-se registros de aproximadamente 1,2 bilhão de partidas e estabeleceu-se métodos para filtrar e pré-processar esses registros, de maneira a torná-los mais adequados a serem utilizados no treino da rede neural;
- Pesquisou-se quais linguagens ou plataformas ofereceriam melhor suporte ao desenvolvimento de uma rede neural artificial com as características necessárias ao problema;
- Criou-se uma rede neural artificial simples em cada uma das linguagens e plataformas identificadas, a fim de estimar qual tornaria o desenvolvimento mais viável e ágil;
- Escolhida a melhor plataforma, implementou-se uma rede neural artificial nela que fosse capaz de treiná-la utilizando a base de dados composta pelos registros adquiridos, filtrados e pré-processados;
- Iniciou-se uma extensa busca, tanto na literatura moderna, quanto experimentalmente, pelos parâmetros e configurações que pudessem fornecer melhores qualidade e tempo de convergência para a rede;
- Analisou-se os resultados obtidos, tentando comparar com o resultados de outro trabalho que tinha como objetivo a solução do mesmo problema.

3.1 Ambiente de desenvolvimento e testes

As características mais relevantes do ambiente utilizado para o desenvolvimento do projeto e para a execução do MLP são as seguintes:

- CPU Intel(R) Core(TM) i5-7500 CPU @ 3.40GHz, 4 Core(s), 4 Logical Processor(s);
- GPU AMD RX 480 4GB GDDR5 @ 1291 MHz;
- Driver de GPU versão 18.10.2;
- SSD 240GB;
- HD 2TB;
- Sistema Operacional Microsoft Windows 10 Home, Versão 10.0.17134 Build 17134;
- Memória RAM física instalada: 16.0 GB DDR4-2134;
- Python versão 3.6.6;
- Keras versão 2.2.2;
- PlaidML versão 0.3.5;
- MATLAB 8.5 versão R2015a;
- NumPy versão 1.14.5.

3.2 Base de dados

Esta seção descreve os passos necessários para a a construção da base de dados que a MLP utilizará para treino, sendo dividida nos tópicos: aquisição dos registros, pré-processamento, filtragem e, por fim, inferência de dados complementares.

3.2.1 Aquisição dos registros de partidas

Foram adquiridos registros de 1.191.768.403 partidas de DOTA 2 ocorridas no intervalo entre março de 2011 a março de 2016. Os registros são disponibilizados por (CUI; CHUNG; HANSON-HOLTRY, 2016), através do portal de *torrents* de dados para uso científico (LO; COHEN, 2016), com o intuito de facilitar e fomentar a utilização de técnicas de aprendizado de máquina para o público. Tipicamente, seriam necessários meses para conseguir requisitar essa quantidade de informações pela API oficial (OPENDOTA, 2015).

Os registros são divididos em três arquivos em três *torrents* distintos, cada um representando um *dump* em JSON de uma tabela PostgreSQL.

- O arquivo *matches* compactado possui tamanho igual a 156GB e contém informações à nível de partidas como: data e horário de início e término da partida, servidor utilizado, ID(s) dos jogadores humanos presentes, heróis selecionados em cada time, time vencedor, entre outros;
- O arquivo *player_matches* é o mais massivo e, mesmo em seu estado compactado, ocupa 542GB de espaço. Contém dados sobre cada jogador em uma partida, como quantas vezes morreu ou matou outros, decisões tomadas, quantidade de dinheiro adquirido por minuto, magias selecionadas e afins;

- O terceiro arquivo, *match_skill*, armazenado em apenas 1,72GB, é o mais leve dos três, pois contém apenas duas informações por registro: o ID da partida e um número de 1 a 3, que corresponde ao nível médio de habilidade dos jogadores que participaram daquela partida. É importante mencionar, porém, que estas informações só estão disponíveis para uma parcela pequena de partidas – em torno de 132 milhões.

Este trabalho utilizou somente os arquivos *matches* e *match_skill*, pois eles já fornecem as informações desejadas: heróis escolhidos, time vencedor, duração da partida e nível médio de habilidade dos jogadores presentes. Foi avaliada, porém logo descartada, a possibilidade de utilizar as informações do arquivo *player_matches*, pois o ambiente de desenvolvimento não possui os requisitos necessários para armazená-lo descompactado, uma vez que estimou-se que ele ocupe acima de 4TB de armazenamento.

3.2.2 Pré-processamento

Em contraste ao formato simples dos registros no arquivo *match_skill*, o arquivo *matches* utiliza uma organização bem mais convoluta, disponível para visualização no Anexo A. Este formato de disposição de dados torna necessário realizar um expurgo dos dados que não serão utilizados antes de importar os dados para um *software* de banco de dados.

Para realizar esta tarefa, foi desenvolvido um *script* em Python que processa o arquivo em *buffer* e, utilizando expressões regulares para extrair somente as informações desejadas, gera um arquivo de saída contendo apenas o ID da partida, duração, heróis escolhidos e time vencedor. Neste ponto, o novo arquivo foi importado para um banco de dados MariaDB.

3.2.3 Filtragem

De posse de mais de 1 bilhão de registros ocorridos ao longo de 5 anos, faz-se necessária a filtragem para tornar os dados mais consistentes antes da utilização desta base de dados para treinamento.

Primeiramente, é necessário utilizar partidas que tenham ocorrido durante uma única versão do jogo, visto que em cada versão são feitos ajustes nos heróis para tentar melhorar o equilíbrio do jogo – enfraquecer heróis que estejam muito fortes e fortalecer heróis que estejam apresentando desempenho muito ruim. Analisou-se as versões do jogo (VALVE, 2018b), de forma a selecionar a que teve maior tempo de permanência dentro do intervalo de tempo dos registros adquiridos. A versão escolhida foi a 6.84c, que permaneceu ativa do dia 18 de maio de 2015 até o dia 24 de setembro de 2015. Limitando-se a esta versão, o número de partidas disponíveis para treinamento cai para 139 milhões.

Além disso, é de interesse utilizar apenas partidas de jogadores de alto nível, a fim de tentar minimizar a ocorrência de partidas em que o fator que mais tenha influenciado a vitória de um time seja a inexperiência de algum oponente; visto que, neste cenário, as vantagens propiciadas pela escolha dos personagens se tornam menos impactantes.

Todavia, como apenas uma pequena parcela desses registros contém informações sobre o nível de habilidade dos jogadores, o número de partidas cai novamente, agora para 18 milhões, ao descartar as partidas em que não são oferecidos os dados supracitados.

Estas são as partidas sobre as quais se sabe qual o nível de habilidade dos jogadores nelas presentes. Os níveis são divididos em: *Normal*, *High Skill* e *Very High Skill*. Finalmente, selecionando apenas as partidas em que os jogadores são do nível mais alto, persistem 881.653 partidas viáveis para compor a base de dados.

3.2.4 Inferência de dados complementares

Apesar de redes neurais lidarem muito bem com entradas numéricas, tem-se um problema caso o modelo a ser treinado exija que sejam informadas entradas representando categorias ou entidades abstratas – por exemplo: cachorro, gato, vaca, cavalo –, pois será necessário codificar estas categorias de alguma forma numérica para que a rede neural possa compreendê-las (BROUWER, 2004).

Os registros de partidas fazem referência aos heróis utilizando números de 1 a 112, porém não devem ser utilizados estes números diretamente como entradas em uma RNA. Isto estaria indicando à rede que existe uma hierarquia clara entre os heróis, o que seria equivalente a dizer que o herói de número 3 equivale ao herói de número 2 somado ao de número 1. Por razão similar, também não é recomendado fazer uma codificação binária desses números, pois acarretaria em um viés entre as classes que utilizassem os mesmos *bits* de ativação; por exemplo, o herói de número 2 (0000010) teria uma semelhança artificial com o de número 65 (1000010), pelo simples fato de ativarem o segundo *bit* da sequência.

Para lidar com esse problema, optou-se por extrair informações numéricas de cada uma das categorias e, então, informar à rede neural cada categoria como um conjunto de métricas sobre a mesma.

Para representar cada herói, utilizou-se as taxas médias de vitória desse herói para diferentes tempos de duração de partidas. Além disso, foi informado à rede qual a sinergia que cada herói tem com aliados e qual vantagem ele tem contra inimigos. Para obter estas informações, os registros foram processados no banco de dados MariaDB e ultimamente dispostos em quatro tabelas simples, disponíveis no Apêndice A. O processamento consiste basicamente nos seguintes passos:

- Verifica-se todas as partidas em que um herói A esteve presente em qualquer um dos times e calcula-se qual o percentual destas partidas em que o time que ele estava inserido foi vitorioso. O resultado será a taxa média geral de vitória do herói A;
- Repete-se o processo acima, porém segregando as partidas em diferentes intervalos de duração, de forma a obter uma taxa média de vitória para o herói A para partidas de diferentes durações, visto que os heróis tem desempenhos diferentes em momentos distintos do jogo. Um herói A pode ter 75% de taxa vitória para partidas que têm duração de 19 a 23 minutos e, ao mesmo tempo, ter 40% de taxa de vitória quando a partida dura mais que 71 minutos. Foi utilizado um intervalo de 4 minutos para amostragem, começando a partir de partidas com duração menor ou igual a 15 minutos, até partidas com duração maior ou igual a 71 minutos, totalizando 14 intervalos;
- Verifica-se todas as partidas em que um herói A esteve presente em um time, enquanto um herói B esteve presente em outro time, e calcula-se qual o percentual destas partidas em que o time do herói A foi vitorioso sobre o time do herói B. O resultado é a vantagem crua do herói A sobre o herói B;
- Calcula-se a diferença entre a vantagem crua do herói A sobre o herói B, com as taxas médias gerais de vitória de ambos os heróis. A este resultado é dado o nome de vantagem relativa do herói A sobre o herói B. Esse procedimento é importante, pois a vantagem crua pode ser enganosa. Se, por exemplo, foi encontrado que um time com um herói A tem vitória sobre um time com herói B 55% das vezes, pode-se pensar que o herói A tem vantagem sobre o B. Todavia, se a taxa média geral de vitória do herói A for de 65% e a taxa média geral de vitória do herói B for de 42%, há a indicação

de que, quando dispostos um contra o outro, a performance do herói A cai em 5% e a performance do herói B sobe em 3%, indicando que o herói B é quem possui vantagem sobre o herói A;

- De maneira análoga ao cálculo da vantagem relativa do herói A sobre o herói B, é realizado o cálculo de sinergia do herói A com o herói B. Este número indica quando dois heróis juntos apresentam um desempenho diferente do esperado, caso sejam analisados individualmente. Exemplo: se um herói A tem taxa média geral de vitória igual a 50% e um herói B tem taxa média geral de vitória igual a 48%, porém, quando um time possui ambos, a taxa sobe para 60%, os heróis A e B possuem uma relação sinergística positiva.

Estes processos foram realizados para todos os heróis, assim como todas as combinações 2-a-2 possíveis de heróis, sendo necessárias algumas horas para concluir. Os códigos SQL utilizados estão disponíveis no Apêndice A.

Utilizando essas informações adicionais, as entradas da rede ficaram configuradas da seguinte forma: herói 1, herói 2, herói 3, (...), e herói 10, em que cada herói corresponde a um conjunto de 25 entradas, totalizando 250 entradas para a rede. As 25 entradas que constituem um herói são:

1 - Taxa média de vitória geral do herói;

2-15 - Taxa média de vitória em partidas de duração em cada um dos intervalos definidos;

16-20 - Vantagem relativa do herói contra cada um dos 5 heróis do time adversário;

21-25 - Sinergia relativa do herói com cada um dos 4 outros integrantes de seu time.

3.3 Codificação da rede MLP

Para determinar em qual plataforma seria implementada a rede, foi realizado um levantamento para identificar quais poderiam fornecer uma boa relação custo-benefício, contrastando primariamente a facilidade da implementação de uma rede MLP em dada plataforma com a velocidade esperada de execução da rede implementada. Também considerou-se a curva de aprendizagem da plataforma, associada ao conhecimento prévio do autor sobre esta. Sob estes parâmetros, os nomes que mais se destacaram foram:

- MATLAB ou sua contrapartida gratuita, Octave;
- Python e sua biblioteca *NumPy*;
- Keras, um *framework* especializado para redes neurais, e sua *engine*, TensorFlow.

Inicialmente, imaginou-se que poderia ser feita a implementação da rede MLP proposta em MATLAB ou Octave, visto que são ambientes de computação numérica que oferecem muitas facilidades para codificar as operações matemáticas que majoritariamente compõem uma rede MLP – além de terem códigos otimizados em *Assembly* para operações, como multiplicações de matriz. Esses fatores motivaram a hipótese de que, devido a estas otimizações e facilidades, uma implementação em MATLAB seria mais rápida em implementação e execução, se comparada a uma implementação feita em uma linguagem de programação *general purpose*.

3.3.1 Sondagem de performance e implementação

Inicialmente, para fins de referência e comparação, foi implementada uma MLP simples de duas camadas em cada uma das três plataformas. Utilizou-se o mesmo pseudocódigo para fazer a implementação em MATLAB e em Python, enquanto no Keras implementou-se uma rede com configuração equivalente.

A base de dados utilizada para testar estas redes foi a *Iris Data Set* (FISHER, 1936), visto que esta é possivelmente a base de dados mais encontrada na literatura de reconhecimento de padrões (DHEERU; TANISKIDOU, 1988). A base contém 3 classes, com 50 instâncias cada, em que cada classe se refere a um tipo da planta iris. Uma classe é linearmente separável das outras duas, não sendo estas últimas linearmente separáveis entre si.

A rede MLP a ser treinada utilizando esta base foi construída no seguinte modelo:

- Entradas: Os 4 atributos presente nas amostras e um *bias*;
- Camada intermediária: 4 neurônios, com função de ativação sigmoide;
- Camada de saída: 3 neurônios, com função de ativação sigmoide;
- Otimizador: SGD;
- Taxa de aprendizado: 0.06;
- Épocas de treino: 500.

3.3.1.1 Velocidade de treino para base Iris

O desempenho temporal foi o ponto que mais segregou as plataformas, favorecendo a implementação em Python. Enquanto o MATLAB precisou de 92,15 segundos para concluir as 500 épocas, a implementação em Python fez o mesmo em apenas 7,24 segundos. Curiosamente, no meio termo ficou o Keras, executando o treino em 37,93 segundos. A Tabela 1 mostra o tempo gasto por amostra e por época em cada implementação.

Possivelmente todas as três implementações poderias ser aceleradas, caso fosse dedicado tempo suficiente otimizando os respectivos códigos. Porém, tendo em vista o maior grau de dificuldade necessário para implementar e configurar uma rede mais complexa, aliado à grande disparidade de desempenho entre o MATLAB e os demais, optou-se por não dar continuidade ao desenvolvimento em MATLAB.

Tabela 1 – Tempo *vs* Plataforma, para Base Iris

	Tempo por treino (milissegundos)	Tempo por amostra (milissegundos)
Python	14,4	0,096
MATLAB	184,0	1,226
Keras	75,8	0,505

3.3.2 Teste de treino com a base de dados própria

Inicialmente, para investigar o desempenho temporal do treinamento com a base de dados proposta, foram feitos testes utilizando uma pequena parcela amostral de 500 amostras aleatórias, dentre as 800.000 disponíveis. Foi implementada uma MLP nas plataformas Keras e Python com as seguintes características:

- Entradas: 250 *datapoints* de uma amostra da base de dados e um bias;
- Camada intermediária: 125 neurônios, com função de ativação sigmoide;
- Camada de saída: 1 neurônio, com função de ativação sigmoide;
- Otimizador: SGD;
- Taxa de aprendizado: 0.06;
- Épocas de treino: 500.

3.3.2.1 Python

Ao treinar o modelo da rede MLP implementado em Python com as 500 amostras, o desempenho foi proibitivamente lento. Para executar cada época de treino, utilizando apenas as 500 amostras da base de dados, eram necessários aproximadamente 10 segundos. Considerando que a proposta é a utilização de 800.000 amostras, cada época de treino iria requerer 4 horas para finalizar. Em treinamentos de MLP, é comum serem necessárias centenas ou milhares épocas de treinamento. A um custo de 4 horas por época, um treinamento hipotético com 3.000 épocas de treino necessitaria de mais de um ano para concluir, o que seria impraticável.

Diante desse cenário, foram dedicados esforços para otimizar o código de diversas maneiras diferentes. Foi feito um preciso *profiling* no código, medindo o tempo exato gasto durante o funcionamento da execução de cada linha. Isso permitiu identificar os gargalos e facilitou a compreensão, bem como a subsequente pesquisa por soluções. As mudanças mais significativas foram:

- Alguns *loops* puderam ser substituídos por operações nativas e otimizadas do NumPy;
- Em certas partes do código, sacrificou-se legibilidade a troco de performance;
- Partes mais custosas do processamento foram isoladas em funções e posteriormente compiladas *Just-in-Time (JIT)*, utilizando o pacote *Numba*.

Com estas modificações, foi possível acelerar o processamento até um ponto plausível: 0,28 segundo para realizar uma época de treino, significando uma melhora de aproximadamente 35 vezes em relação ao código anterior. Para realizar 3.000 épocas de treino utilizando 800.000 amostras, seriam necessárias 342 horas, ou aproximadamente 14 dias – ainda lento, porém viável.

Outras abordagens que também foram consideradas, mas não resultaram em êxito, podem ser citadas:

- Paralelização de certas partes do processamento;

- Declaração tipada e compilação *Ahead-Of-Time* (AOT).

Contudo, utilizando de mais ajustes no processo da compilação JIT, foi possível dobrar a velocidade de execução anterior, melhorando outra vez o desempenho. Neste ponto, era gasto 0,135 segundo por época de treino. Nesse cenário, para realizar as mesmas 3.000 épocas de treino com 800.000 amostras, seriam necessárias 180 horas - aproximadamente uma semana.

3.3.2.2 Keras

Foi implementada no Keras uma rede com estrutura equivalente à rede implementada manualmente em Python. Para treinar uma época nessa rede sob as mesmas condições, ela requer 0,299 segundo – aproximadamente o mesmo tempo que era gasto pela rede em Python antes da última etapa de otimização.

Entretanto, apesar de em princípio sua velocidade de treino aparentar aquém da alternativa, em termos práticos, o Keras é capaz de fornecer não só mais rapidez no tempo de processamento por época, mas agilizar o desenvolvimento como um todo. Por exemplo, utilizando o modo de treinamento em *batch*, ele foi capaz de realizar o treino de uma época com 500 amostras em 0,0029 segundo, com a condição de uma memória RAM suficiente disponível no sistema. E a utilização desta funcionalidade requer pouco mais que o ajuste de uma linha de código. Outras facilidades incluem: diversas funções de otimização e de ativação disponíveis, técnicas para tentar evitar *overfitting* ou ainda técnicas para acelerar a convergência.

Possivelmente, fazendo a implementação do processamento em *batch* no código à mão em Python, ele novamente tomaria a liderança ou ficaria tão rápido quanto o código implementado no Keras. Porém, isso viria a um grande custo de tempo de implementação. Em termos práticos, o Keras torna imensamente mais simples de variar completamente todo e qualquer tipo de parâmetro da rede e fornece inúmeras funcionalidades de forma acessível, facilitando a busca por parâmetros e hiperparâmetros mais adequados que possam melhorar o desempenho geral da rede.

Considerados esses fatores, optou-se por utilizar o Keras como plataforma de implementação da rede MLP final, pois ele reduziria o tempo entre a ideia e implementação e permitiria uma investigação mais eficiente sobre o comportamento da MLP. Um estudo em que, ao invés de precisar desprender um enorme esforço com aprendizado sobre peculiaridades de codificação em uma linguagem específica, a ênfase poderia ficar no aprendizado sobre como diferentes técnicas de redes neurais se relacionam entre si e em como as características da rede afetam o comportamento e convergência da mesma.

3.4 Otimização dos parâmetros e hiperparâmetros

Cada rede neural artificial é constituída por uma série de parâmetros e configurações que a definem. Exemplos de parâmetros que podem ser citados são: a quantidade de camadas, o número de neurônios por camada, as taxas de aprendizado, a função de ativação particular dos neurônios em cada camada, a função otimizadora que irá reger o processo de atualização dos pesos ou ainda parâmetros de técnicas mais recentes como *dropout*. A combinação destes parâmetros define fundamentalmente o comportamento e capacidade de aprendizado de uma rede neural artificial.

A quantidade de combinações que podem ser feitas com esses parâmetros é virtualmente infinita, pois enquanto alguns parâmetros estão limitados a N configurações diferentes, outros – como a quantidade de camadas da rede e a largura de cada camada – são limitados apenas pela quantidade de memória e processamento disponível no sistema. Sendo assim, um dos maiores desafios no desenvolvimento de uma rede neural artificial é determinar a melhor estrutura e os parâmetros da rede para o treinamento do problema (XU; CHEN, 2008) (SNOEK; LAROCHELLE; ADAMS, 2012) (DIAZ et al., 2017).

Tipicamente, esses parâmetros são escolhidos baseando-se em regras heurísticas e, logo após, são refinados de forma manual. Este processo requer muito tempo, pois podem ser necessárias vários horas para se avaliar a performance de um único conjunto de parâmetros para a rede neural artificial (DIAZ et al., 2017). Além disso, "*infelizmente, o afinamento é frequentemente uma 'arte obscura', requerindo experiência profissional, regras práticas ou, às vezes, tentativa e erro*" [1](SNOEK; LAROCHELLE; ADAMS, 2012, tradução nossa).

Para definir inicialmente os parâmetros a serem utilizados durante o treinamento da RNA desenvolvida, foram utilizadas uma combinação de regras heurísticas tradicionais e recomendações modernas da área, provenientes dos exemplos mais marcantes e bem sucedidos que surgiram nos últimos 6 anos.

3.4.1 Número de camadas inicial

Se tratando de RNAs, a utilização de uma única camada intermediária já é capaz de modelar até as funções mais complexas, desde que possua neurônios suficientes. Diante disso, uma abordagem sugerida é que se faça o treinamento com apenas uma ou duas camadas intermediárias e, caso necessário, aumente gradativamente o número de camadas até começar a ocorrer *overfitting*. Uma outra regra prática interessante de se observar diz que: em geral, aumentar o número de camadas é mais vantajoso que aumentar o número de neurônios por camada, pois costuma propiciar uma melhor relação custo-benefício em termos de grau de convergência por tempo gasto (GÉRON, 2017, p. 275). Seguindo estas recomendações, a topologia inicial utilizou duas camadas intermediárias, além das camadas de entrada e saída indispensáveis.

3.4.2 Quantidade de neurônios inicial

A quantidade de neurônios nas camadas de entrada e de saída é determinada naturalmente pelo formato dos registros na base de dados que a RNA utilizará para treinar. A base de dados utilizada possui duzentas e cinquenta entradas e uma saída. Quanto às camadas intermediárias, uma prática comum é defini-las com uma quantidade progressivamente menor de neurônios em cada camada, estabelecendo um formato afunilado no sentido da camada de saída. A racionalização dessa estratégia baseia-se no fato de que características de baixo nível – geralmente extraídas pelas primeiras camadas – coalescem numa quantidade bem menor de características de alto nível, que são tradadas pelas camadas finais (GÉRON, 2017, p. 276). Baseando-se nessas considerações, utilizou-se: 250, 200, 50 e 1 neurônios nas camadas inicial, intermediárias e final, respectivamente. A Figura <figura> ilustra esta topologia.

[1] "*Unfortunately, this tuning is often a 'blackart' requiring expert experience, rules of thumb, or sometimes brute-force search.*"

4 Resultados

4.0.1 Topologia inicial adotada

Embora a literatura recente endosse a utilização da função de ativação ReLU para a maior parte dos casos (GÉRON, 2017, p. 340), pressupõe-se que ela não seria a alternativa mais adequada para treinar a base de dados proposta. A racionalização dessa hipótese é baseada no fato da função ReLU mapear qualquer entrada negativa em zero. Tendo em vista que as entradas da base de dados podem conter diversos valores negativos – indicando as vantagens e sinergias que cada herói possui em relação a outro –, utilizar a função ReLU possivelmente impediria o desenvolvimento de neurônios especializados em desvantagens.

Com base nas características das funções de ativação descritas na seção 2.1.7, acredita-se que o mais interessante seja utilizar a função tangente hiperbólica para as camadas intermediárias e a função sigmoidal para a camada de saída. Todavia, como seria imprudente desconsiderar todos os bons endossamentos que as funções retificadoras têm recebido, optou-se por avaliar também a função ReLU como função de ativação das camadas intermediárias.

4.1 Testes de parâmetros

A partir desses parâmetros iniciais, foram realizados acima de quatrocentos testes distintos de treinamento, realizando pequenas variações em um parâmetro de cada vez.

Os primeiros testes foram realizados utilizando bases amostrais pequenas, a fim de nortear qual parâmetro mais influenciaria na velocidade e investigá-lo o quanto antes. Isto é importante, pois os testes utilizando a base de dados integral podem ser demorados. O parâmetro que mais aparentava afetar a velocidade de convergência era a escolha do método otimizador e, portanto, este foi o primeiro a ser investigado também para a base completa.

4.1.1 Teste de otimizadores

Para identificar qual algoritmo otimizador apresenta o melhor desempenho, foram realizados 20 treinos de teste em 5 cenários diferentes, em que foram variadas as funções otimizadoras, as funções de ativação das camadas intermediárias e a utilização do método *Dropout*, mantendo as seguintes configurações em comum:

- Quantidade máxima de épocas de treino: 3.000;
- Quantidade de amostras na base de dados: 800.000;
- Parcela da base de dados reservada para teste: 20%;
- Quantidade de camadas intermediárias: 2;
- Quantidades de neurônios nas camadas intermediárias e final: 200, 50, e 1, respectivamente.

Os cenários foram divididos com o objetivo de verificar se configurações diferentes da rede podem interferir na escolha de um otimizador. Abaixo são apresentados 4 cenários levantados.

- Cenário A — Utiliza tangente hiperbólica, sem *Dropout*;
- Cenário B — Utiliza tangente hiperbólica, com *Dropout*;
- Cenário C — Utiliza ReLU, sem *Dropout*;
- Cenário D — Utiliza ReLU, com *Dropout*.

As métricas mais relevantes dos treinos em cada cenário foram sumarizadas nas Tabelas de número 2, 3, 4 e 5. Pode-se observar que nessas condições, o otimizador Adam demonstrou desempenho superior a todos os outros em três dos quatro cenários. Sua precisão máxima alcançada na base de teste foi superior, enquanto o tempo gasto e a quantidade de épocas necessárias para alcançar a precisão foram menores. Pode-se ressaltar ainda que, no cenário B, o único no qual ele não lidera em todos os pontos, tem-se um desempenho praticamente idêntico ao líder no cenário B, o otimizador Nadam.

Outro detalhe interessante a ser destacado é a diferença de desempenho de alguns otimizadores quando é utilizada a função de ativação ReLU. O Adagrad continuou o treino por mais tempo e atingiu uma precisão maior nos cenários em que foi utilizada a função ReLU, em comparação aos que utilizaram a função tangente hiperbólica. Em contrapartida, os otimizadores Adam, RMSProp e Nadam convergiram mais rápido, porém estes dois últimos apresentaram uma redução na precisão máxima alcançada. O otimizador SGD, por sua vez, possui uma velocidade de convergência tão baixa que não foi possível observar mudanças significativas em nenhum dos cenários avaliados em 3.000 épocas de treino.

Tabela 2 – Métricas observadas sobre o cenário A. Função de ativação Tangente hiperbólica, **sem** utilização de *Dropout*

Métrica	**SGD**	**Adam**	**AdaGrad**	**RMSProp**	**Nadam**
Tempo gasto (hh:mm:ss)	1:34:14	0:23:08	0:38:51	0:44:43	0:27:58
Quantidade de épocas treinadas	3000	733	1236	1427	876
Precisão máxima na base de teste	0.60833	0.61654	0.61265	0.61061	0.61441
Época da máx. precisão na base de teste	2976	593	1178	1211	660

Fonte – Elaborada pelo autor.

Tabela 3 – Métricas observadas sobre o teste B. Função de ativação Tangente hiperbólica, **com** utilização de *Dropout*

Métrica	**SGD**	**Adam**	**AdaGrad**	**RMSProp**	**Nadam**
Tempo gasto (hh:mm:ss)	1:34:27	0:39:52	0:54:13	0:43:31	0:36:12
Quantidade de épocas treinadas	3000	1261	1725	1377	1163
Precisão máxima na base de teste	0.60510	0.61564	0.61318	0.61280	0.61524
Época da máx. precisão na base de teste	2992	1070	1678	1161	1067

Fonte – Elaborada pelo autor.

Tabela 4 – Métricas observadas sobre o teste C. Função de ativação ReLU, **sem** utilização de *Dropout*

Métrica	**SGD**	**Adam**	**AdaGrad**	**RMSProp**	**Nadam**
Tempo gasto (hh:mm:ss)	1:35:42	0:25:20	1:37:05	0:37:05	0:28:25
Quantidade de épocas treinadas	3000	767	3000	933	835
Precisão máxima na base de teste	0.60651	0.61527	0.61436	0.60769	0.60971
Época da máx. precisão na base de teste	2959	302	2855	468	370

Fonte – Elaborada pelo autor.

Tabela 5 – Métricas observadas sobre o teste D. Função de ativação ReLU, **com** utilização de *Dropout*

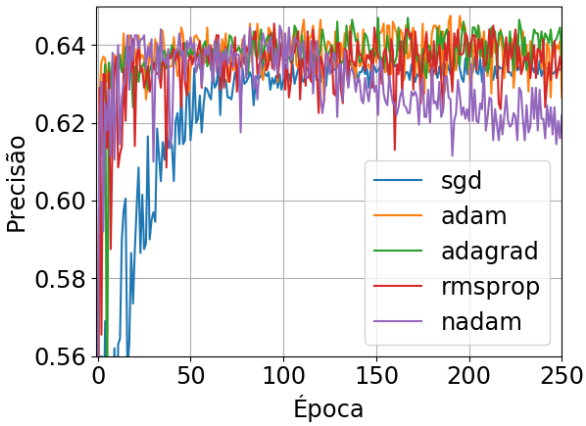
Métrica	**SGD**	**Adam**	**AdaGrad**	**RMSProp**	**Nadam**
Tempo gasto (hh:mm:ss)	1:37:08	0:34:19	1:36:34	1:25:57	0:41:50
Quantidade de épocas treinadas	3000	1022	3000	2687	1307
Precisão máxima na base de teste	0.60357	0.61443	0.61397	0.61086	0.61284
Época da máx. precisão na base de teste	2992	557	2937	2239	842

Fonte – Elaborada pelo autor.

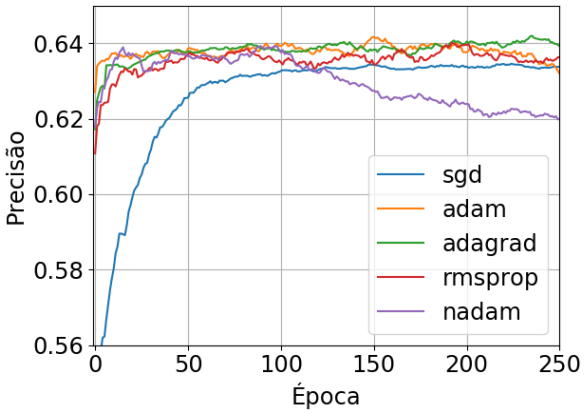
Os resultados contrastando os diferentes otimizadores em cada cenário também são apresentados nas Figuras de número 9 a 19, na forma de gráficos que mostram a precisão atingida em função da época. As duas primeiras figuras para cada cenário mostram os resultados combinados de todos os otimizadores avaliados, enquanto a figura posterior apresenta individualmente o comportamento de cada otimizador. Através das figuras, é possível fazer outras observações, como a influência do *dropout* em manter as precisões nas bases de treino e teste mais próximas.

Para permitir uma boa visibilidade, os gráficos tiveram que ser amortecidos utilizando a técnica de média móvel (SMITH et al., 1997, p. 277), caso contrário as variações na saída do sinal tornariam impossível a demonstração de dois ou mais sinais no mesmo gráfico, devido à sobreposição em excesso. A janela de amortecimento utilizada foi de 15 pontos. A Figura 8 exemplifica esse efeito exibindo dois gráficos plotados empregando os mesmos dados e nas mesmas escalas, porém um deles utilizando os dados crus e o outro, os dados amortecidos pela média móvel.

Figura 8 – Exemplo do efeito da utilização da Média Móvel para amortecimento dos dados que serão apresentados



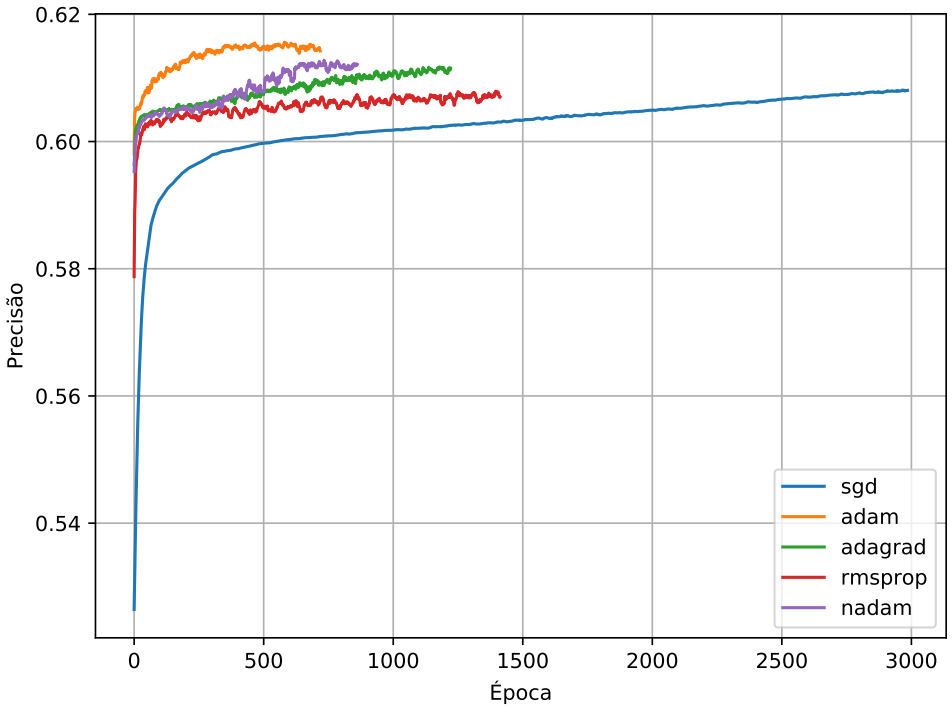
(a) Dados plotados diretamente



(b) Dados plotados amortecidos por Média Móvel com janela de amortecimento n = 15

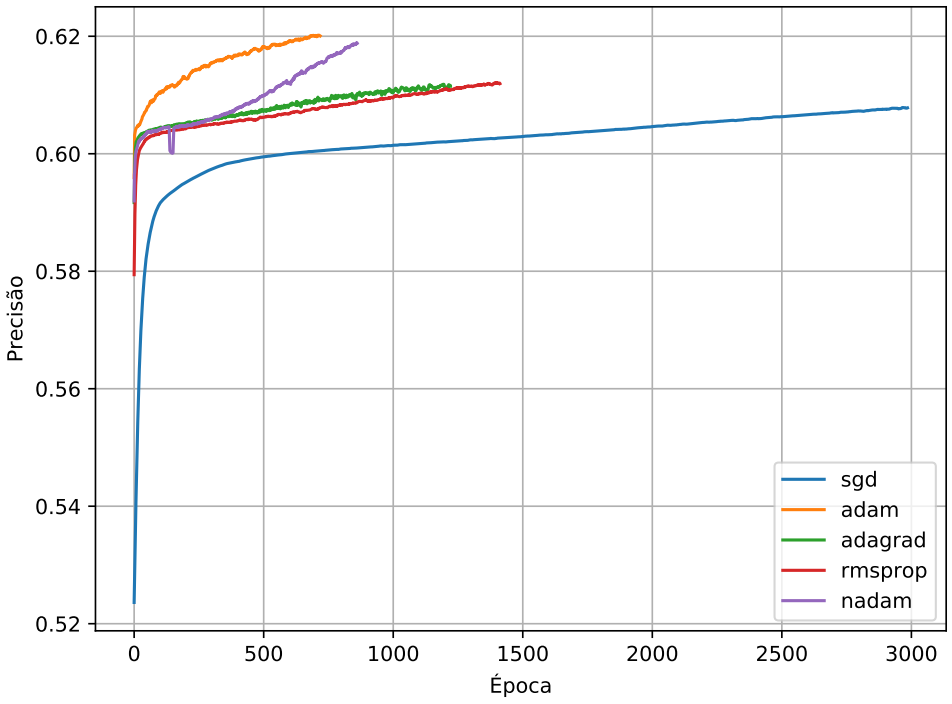
Fonte – Elaborada pelo autor.

Figura 9 – Cenário A - Precisão *versus* época na base de **teste**, utilizando diferentes otimizadores. Função de ativação tangente hiperbólica, **sem** utilização de *Dropout*



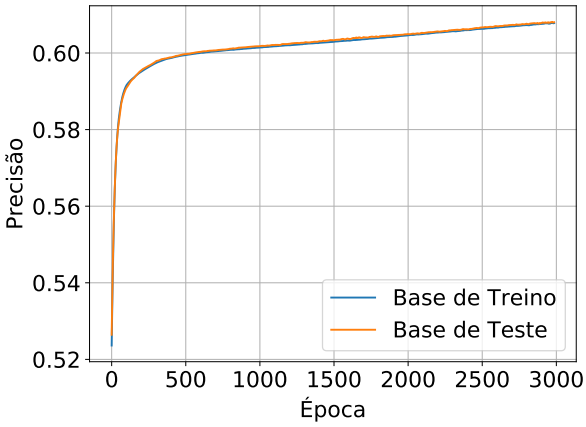
Fonte – Elaborada pelo autor.

Figura 10 – Cenário A - Precisão *versus* época na base de **treino**, utilizando diferentes otimizadores. Função de ativação tangente hiperbólica, **sem** utilização de *Dropout*.

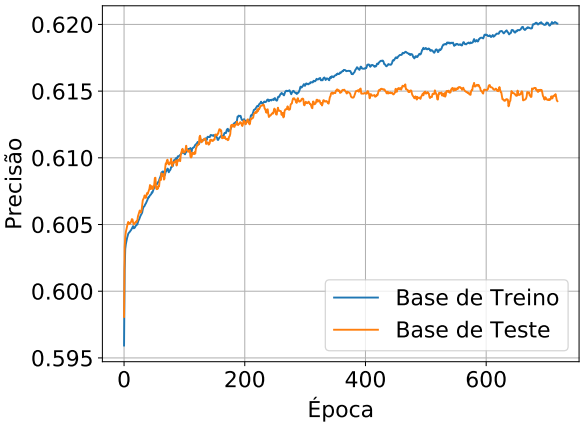


Fonte – Elaborada pelo autor.

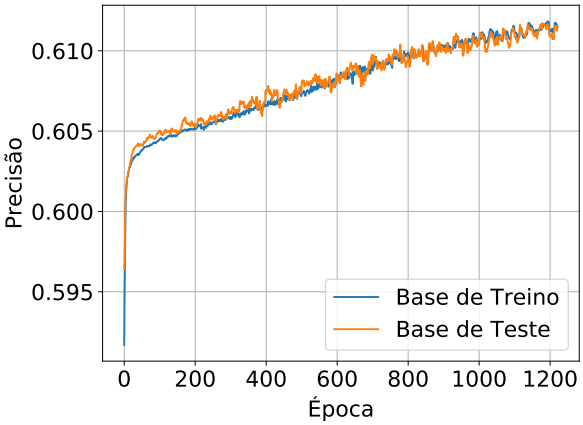
Figura 11 – Comportamento individual dos treinamentos realizados no Cenário A. Função de ativação Tangente hiperbólica, **sem** utilização de *Dropout*



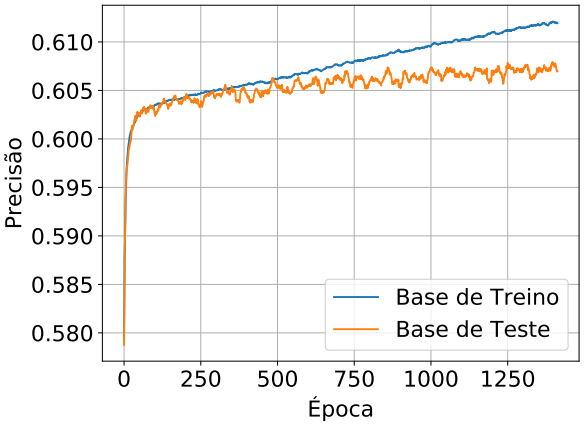
(a) Utilizando otimizador SGD



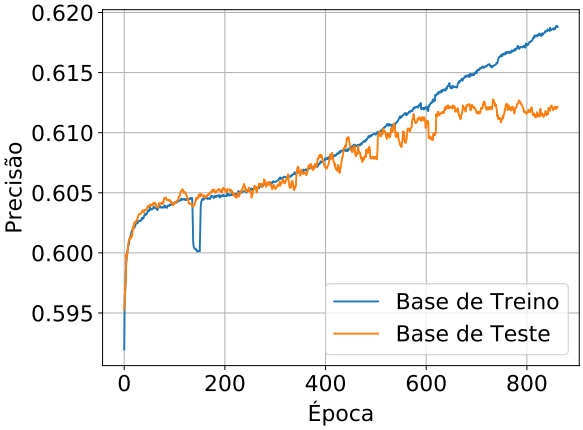
(b) Utilizando otimizador Adam



(c) Utilizando otimizador AdaGrad



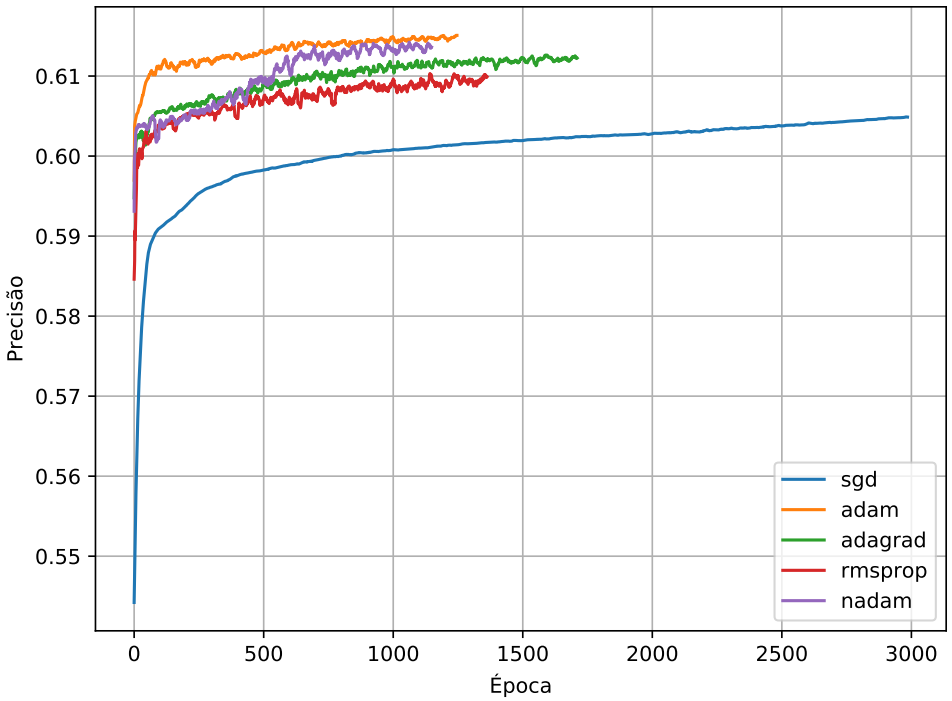
(d) Utilizando otimizador RMSProp



(e) Utilizando otimizador Nadam

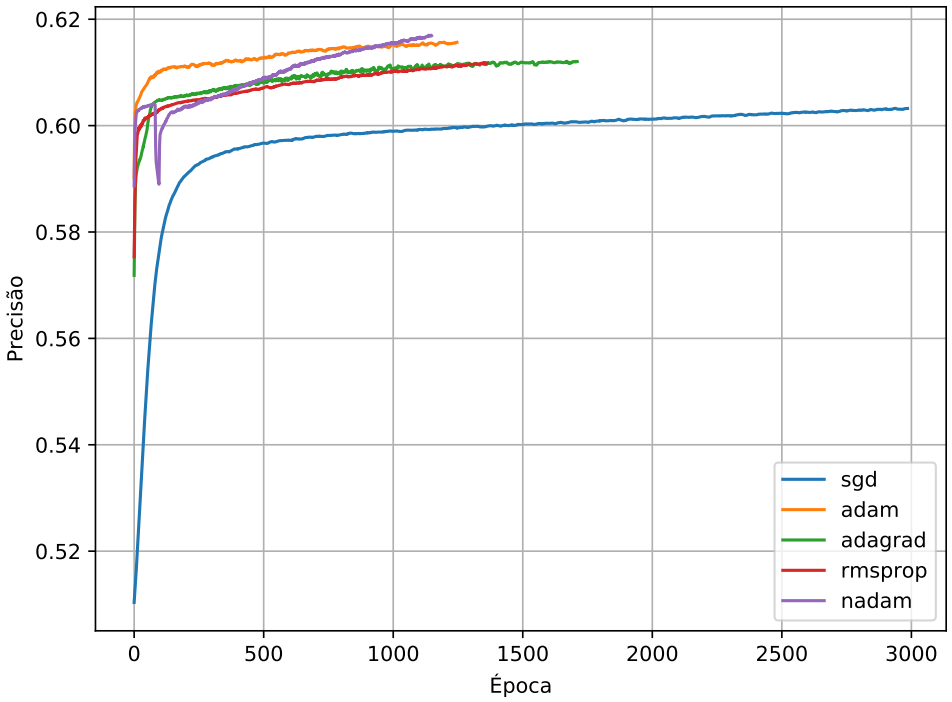
Fonte – Elaborada pelo autor.

Figura 12 – Cenário B - Precisão *versus* época na base de **teste**, utilizando diferentes otimizadores. Função de ativação Tangente hiperbólica, **com** utilização de *Dropout*



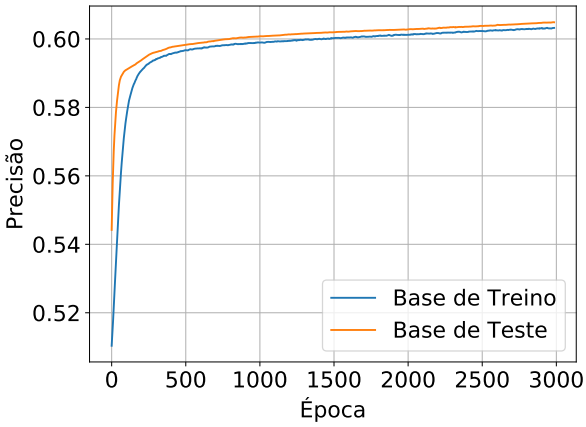
Fonte – Elaborada pelo autor.

Figura 13 – Cenário B - Precisão *versus* época na base de **treino**, utilizando diferentes otimizadores. Função de ativação Tangente hiperbólica, **com** utilização de *Dropout*

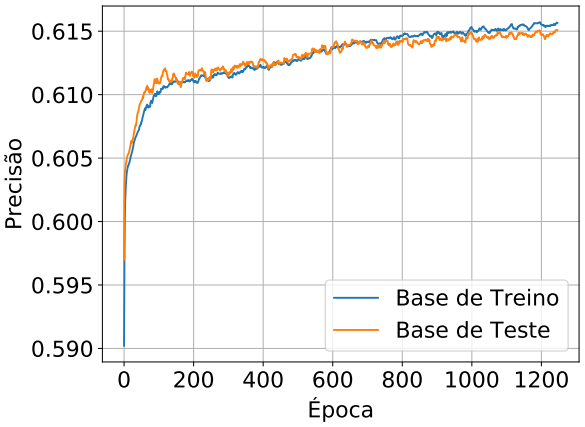


Fonte – Elaborada pelo autor.

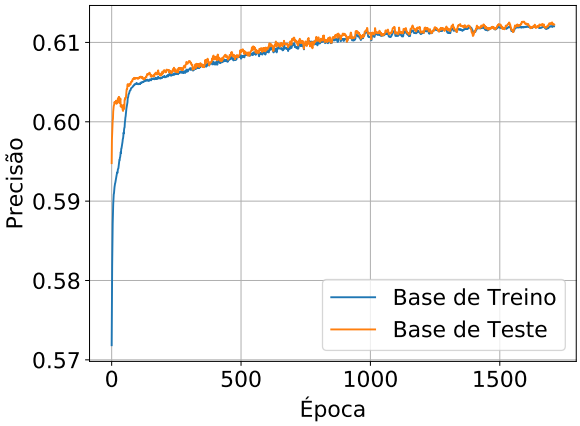
Figura 14 – Comportamento individual dos treinamentos realizados no Cenário B. Função de ativação Tangente hiperbólica, **com** utilização de *Dropout*



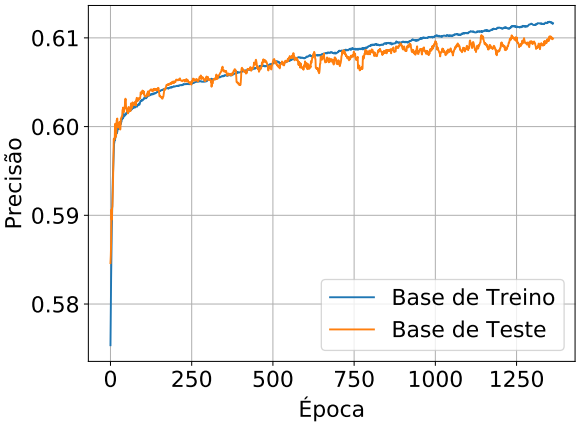
(a) Utilizando otimizador SGD



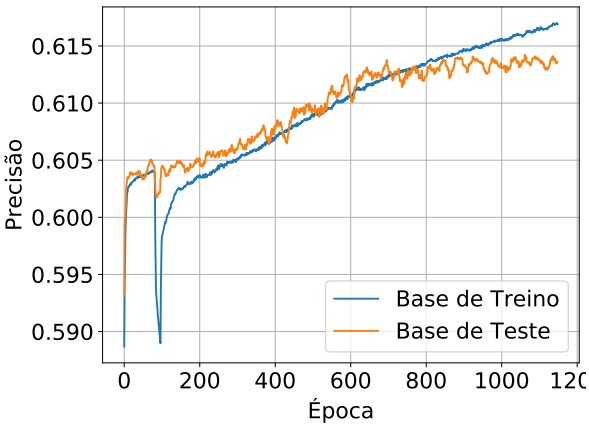
(b) Utilizando otimizador Adam



(c) Utilizando otimizador AdaGrad

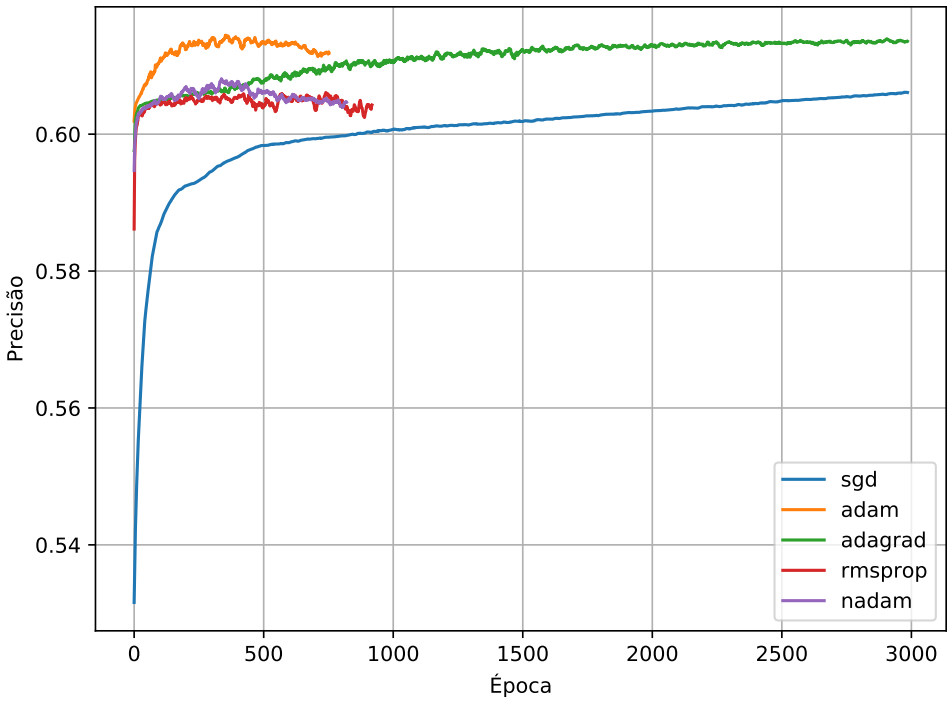


(d) Utilizando otimizador RMSProp



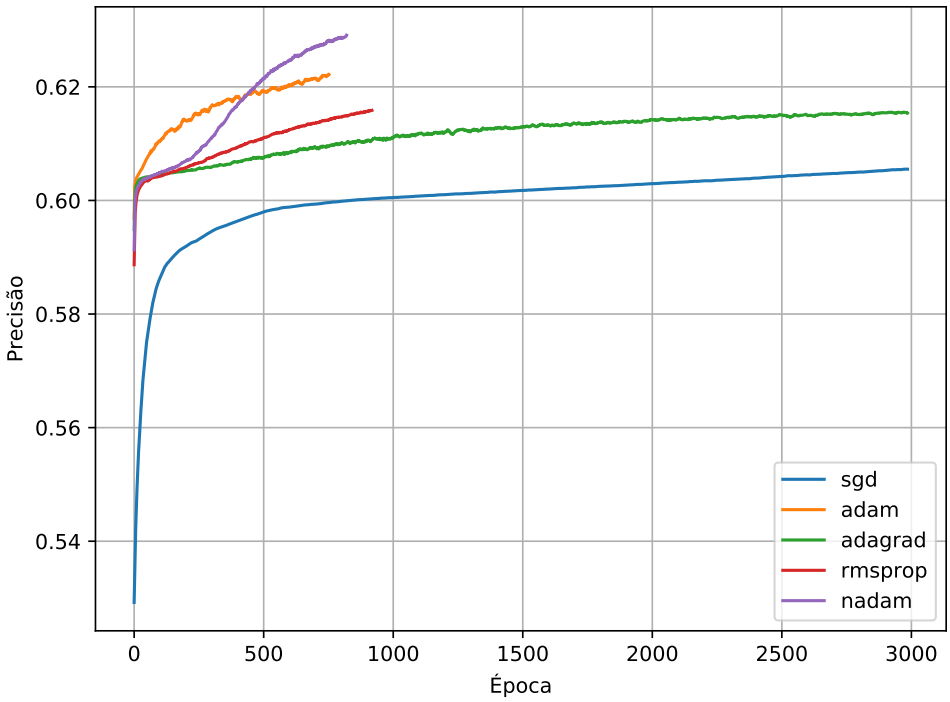
(e) Utilizando otimizador Nadam

Figura 15 – Cenário C - Precisão *versus* época na base de **teste**, utilizando diferentes otimizadores. Função de ativação ReLU, **sem** utilização de *Dropout*



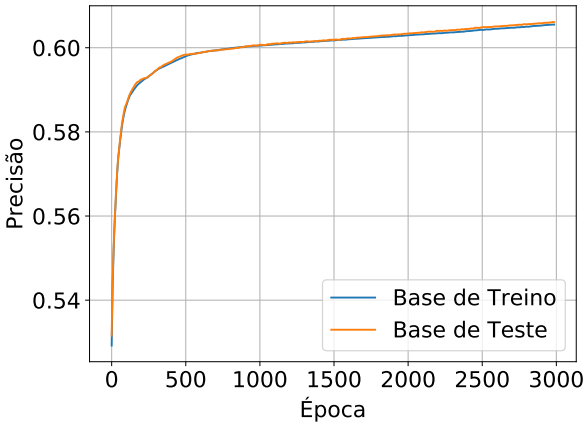
Fonte – Elaborada pelo autor.

Figura 16 – Cenário C - Precisão *versus* época na base de **treino**, utilizando diferentes otimizadores. Função de ativação ReLU, **sem** utilização de *Dropout*

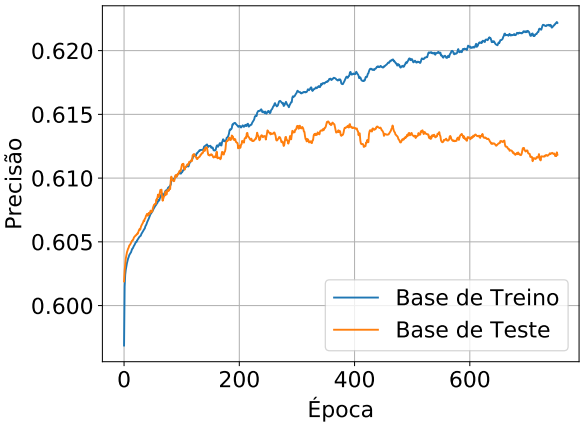


Fonte – Elaborada pelo autor.

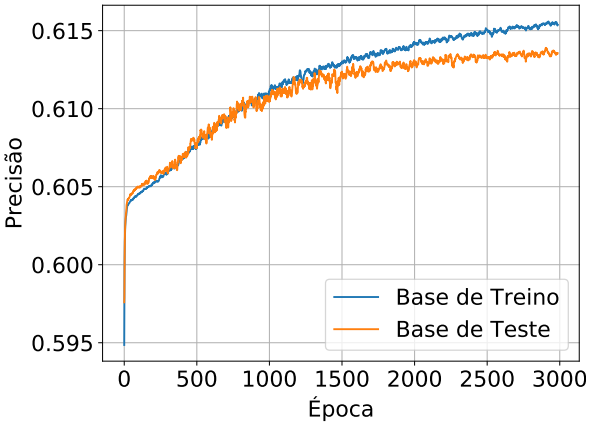
Figura 17 – Comportamento individual dos treinamentos realizados no Cenário C. Função de ativação ReLU, **sem** utilização de *Dropout*



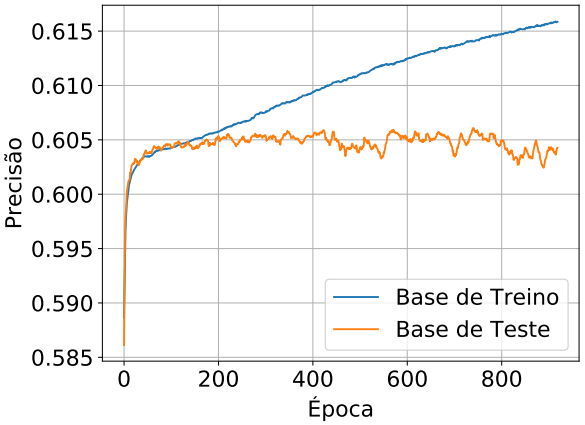
(a) Utilizando otimizador SGD



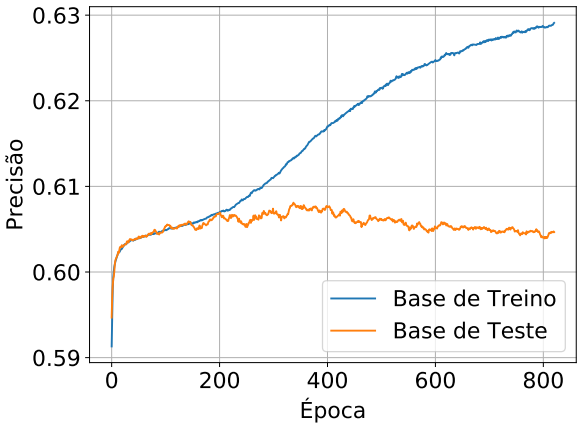
(b) Utilizando otimizador Adam



(c) Utilizando otimizador AdaGrad



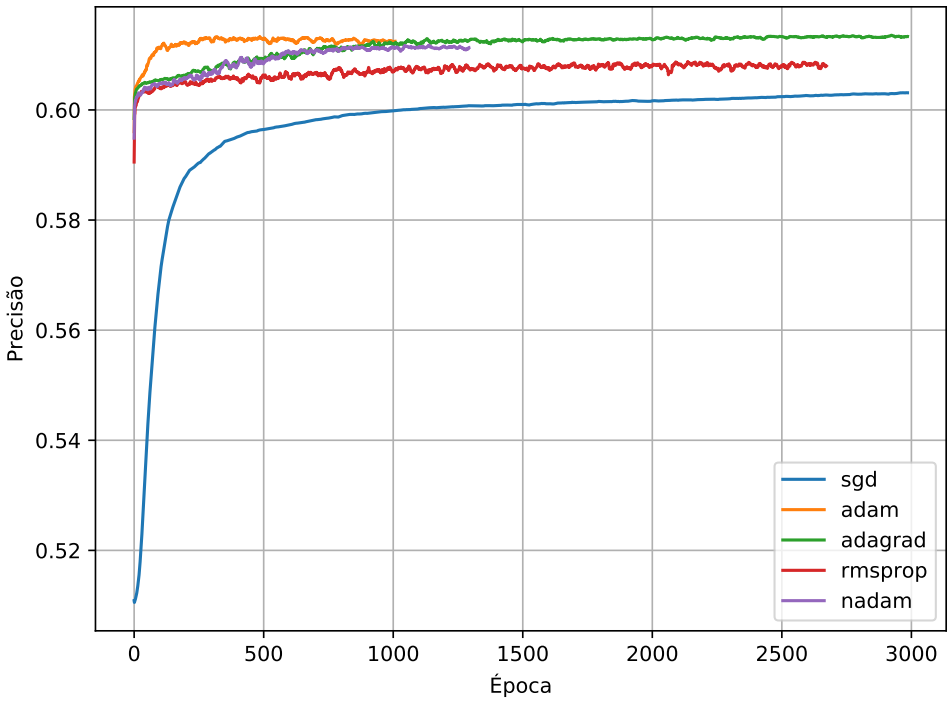
(d) Utilizando otimizador RMSProp



(e) Utilizando otimizador Nadam

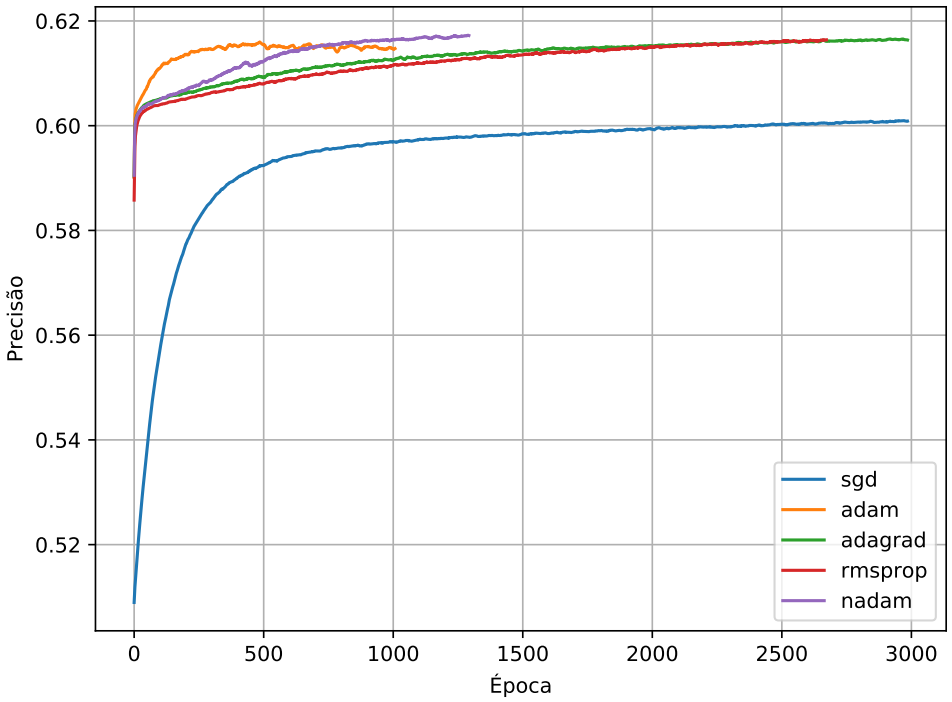
Fonte – Elaborada pelo autor.

Figura 18 – Cenário D - Precisão *versus* época na base de **teste**, utilizando diferentes otimizadores. Função de ativação ReLU, **com** utilização de *Dropout*



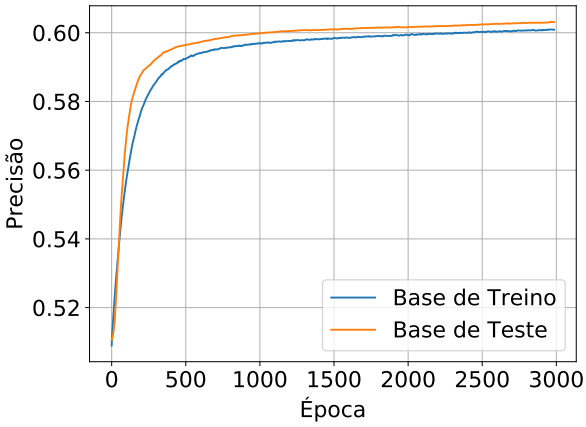
Fonte – Elaborada pelo autor.

Figura 19 – Cenário D - Precisão *versus* época na base de **treino**, utilizando diferentes otimizadores. Função de ativação ReLU, **com** utilização de *Dropout*

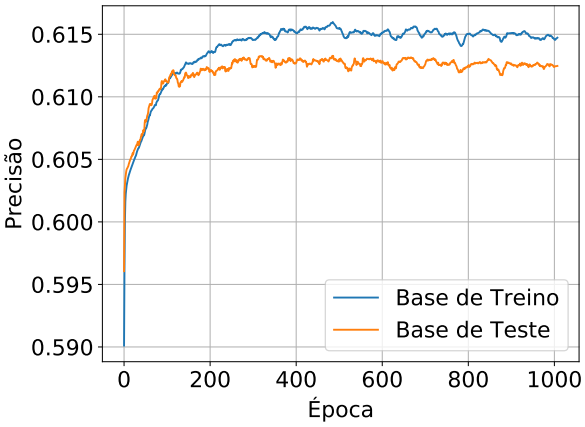


Fonte – Elaborada pelo autor.

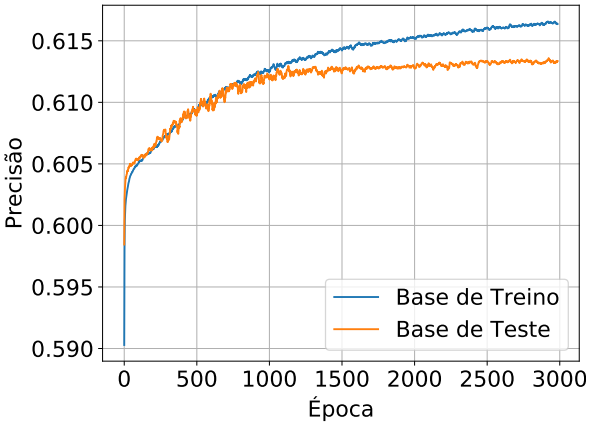
Figura 20 – Comportamento individual dos treinamentos realizados no Cenário D. Função de ativação ReLU, **com** utilização de *Dropout*



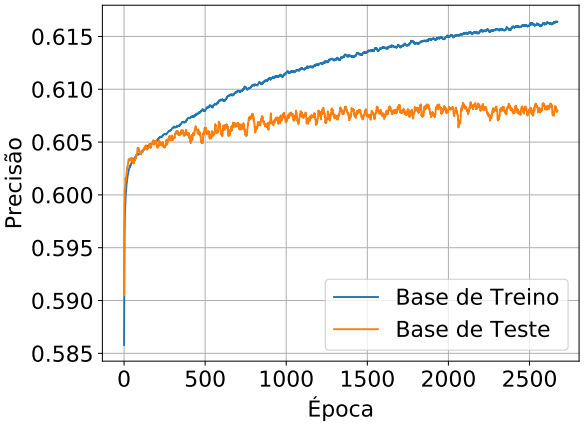
(a) Utilizando otimizador SGD



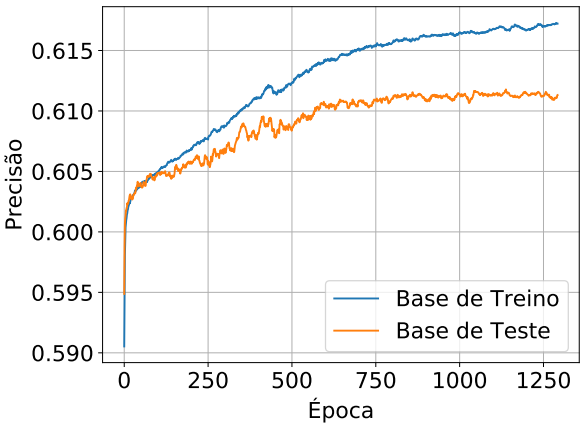
(b) Utilizando otimizador Adam



(c) Utilizando otimizador AdaGrad



(d) Utilizando otimizador RMSProp



(e) Utilizando otimizador Nadam

Fonte – Elaborada pelo autor.

4.1.2 Variação das configurações de camadas

Determinado qual o otimizador a ser utilizado, pôde-se começar a investigar qual configuração de camadas fornece resultados melhores, incluindo a quantidade de camadas, a quantidade de neurônios por camada e a função de ativação dos neurônios. Para isso, foram realizados 30 treinos de teste em 6 cenários diferentes, mantendo as seguintes configurações em comum para todos os cenários:

- Quantidade máxima de épocas de treino: 3.000;
- Quantidade de amostras na base de dados: 800.000;
- Parcela da base de dados reservada para teste: 20%;
- Otimizador Adam.

As configurações investigadas foram divididas nos seguintes cenários:

- Cenário E — Utiliza tangente hiperbólica, sem *Dropout*;
- Cenário F — Utiliza tangente hiperbólica, com *Dropout*;
- Cenário G — Utiliza ReLU, sem *Dropout*;
- Cenário H — Utiliza ReLU, com *Dropout*;
- Cenário I — Utiliza Leaky ReLU, sem *Dropout*;
- Cenário J — Utiliza Leaky ReLU, com *Dropout*.

Assim como no teste variando otimizadores, as métricas mais relevantes dos treinos em cada cenário foram sumarizadas e estão disponíveis nas Tabelas de número 6 a 11. Para facilitar a visualização, a precisão máxima na base de teste para todas as configurações foi plotada no Gráfico 21 e a época em que tal precisão foi atingida foi plotada no Gráfico 22.

Apesar desses resultados serem mais trabalhosos para se analisar, pois não indicam uma configuração que seja universalmente superior às outras, a Figura 21 permite visualizar que, em termos de precisão, a configuração 200-100-50 é dominada tanto pela configuração 200-50, quanto pela 100-25, pois apresenta resultado pior que estas duas em todos os cenários. Por outro lado, a Figura 22 mostra que esta mesma configuração 200-100-50 foi consistentemente uma das mais rápidas a atingir seu pico de precisão. Pode-se argumentar que, como a diferença máxima de precisão atingida entre as configurações é relativamente baixa – em torno de 0,3% –, a escolha da configuração a ser adotada dependerá de qual fator for considerado mais importante, não existindo uma que seja absolutamente melhor que as outras.

Os gráficos contrastando as diferentes configurações de camada também são apresentados para cada cenário nas Figuras de número 23 a 34, mostrando a precisão atingida em função da época. Os gráficos que descrevem o comportamento individual em cada treino estão disponíveis no Apêndice B. Novamente, é possível observar a eficácia do *Dropout* em evitar o *overfit* e manter próximas as precisões entre as bases de treino e teste. Por exemplo, na Figura 31, nota-se que a precisão na base de teste em algumas configurações diminui bastante após atingir seu máximo, enquanto na Figura 33 isto não acontece. A configuração 200-100-50 no cenário I apresenta indicativos claros de *overfitting*.

Tabela 6 – Métricas observadas sobre o teste E. Função de ativação Tangente hiperbólica, **sem** utilização de *Dropout*

Métrica	**200-50**	**200-100-50**	**300-75**	**100-25**	**100-50-25-10**
Tempo gasto (hh:mm:ss)	0:35:17	0:24:16	0:28:50	0:32:24	0:27:27
Quantidade de épocas treinadas	1089	709	868	1059	849
Precisão máx. na base de teste	0.61619	0.61602	0.61593	0.61606	0.61576
Época da máx. precisão	624	244	403	594	384

Fonte – Elaborada pelo autor.

Tabela 7 – Métricas observadas sobre o teste F. Função de ativação Tangente hiperbólica, **com** utilização de *Dropout*

Métrica	**200-50**	**200-100-50**	**300-75**	**100-25**	**100-50-25-10**
Tempo gasto (hh:mm:ss)	1:37:04	0:33:04	1:28:53	1:00:21	0:29:39
Quantidade de épocas treinadas	3000	946	2765	2084	926
Precisão máx. na base de teste	0.61623	0.61396	0.61651	0.61526	0.61339
Época da máx. precisão	2863	758	2654	1910	913

Fonte – Elaborada pelo autor.

Tabela 8 – Métricas observadas sobre o teste G. Função de ativação ReLU, **sem** utilização de *Dropout*

Métrica	**200-50**	**200-100-50**	**300-75**	**100-25**	**100-50-25-10**
Tempo gasto (hh:mm:ss)	0:30:26	0:25:24	0:24:09	0:29:06	0:28:19
Quantidade de épocas treinadas	939	741	748	926	802
Precisão máx. na base de teste	0.61538	0.61504	0.61492	0.61537	0.61510
Época da máx. precisão	474	276	283	461	337

Fonte – Elaborada pelo autor.

Tabela 9 – Métricas observadas sobre o teste H. Função de ativação ReLU, **com** utilização de *Dropout*

Métrica	**200-50**	**200-100-50**	**300-75**	**100-25**	**100-50-25-10**
Tempo gasto (hh:mm:ss)	0:32:24	0:29:59	0:23:24	0:40:27	0:34:53
Quantidade de épocas treinadas	1020	856	712	1261	1044
Precisão máx. na base de teste	0.61491	0.61442	0.61480	0.61462	0.61408
Época da máx. precisão	555	819	247	796	579

Fonte – Elaborada pelo autor.

Tabela 10 – Métricas observadas sobre o teste I. Função de ativação Leaky ReLU, **sem** utilização de *Dropout*

Métrica	**200-50**	**200-100-50**	**300-75**	**100-25**	**100-50-25-10**
Tempo gasto (hh:mm:ss)	0:28:56	0:24:48	0:27:34	0:48:56	0:27:30
Quantidade de épocas treinadas	871	738	836	1530	812
Precisão máx. na base de teste	0.61592	0.61509	0.61581	0.61619	0.61484
Época da máx. precisão	406	273	371	1065	347

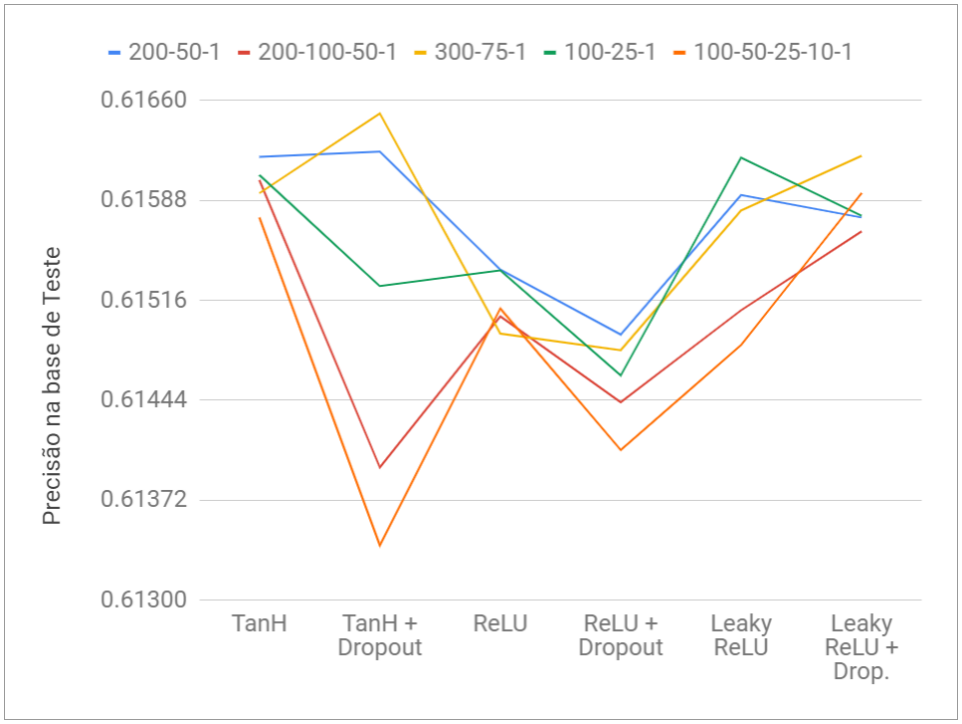
Fonte – Elaborada pelo autor.

Tabela 11 – Métricas observadas sobre o teste J. Função de ativação Leaky ReLU, **com** utilização de *Dropout*

Métrica	**200-50**	**200-100-50**	**300-75**	**100-25**	**100-50-25-10**
Tempo gasto (hh:mm:ss)	0:24:32	0:27:04	0:42:37	0:34:40	0:45:05
Quantidade de épocas treinadas	802	814	1342	1177	1381
Precisão máx. na base de teste	0.61576	0.61566	0.61620	0.61577	0.61593
Época da máx. precisão	337	349	877	712	916

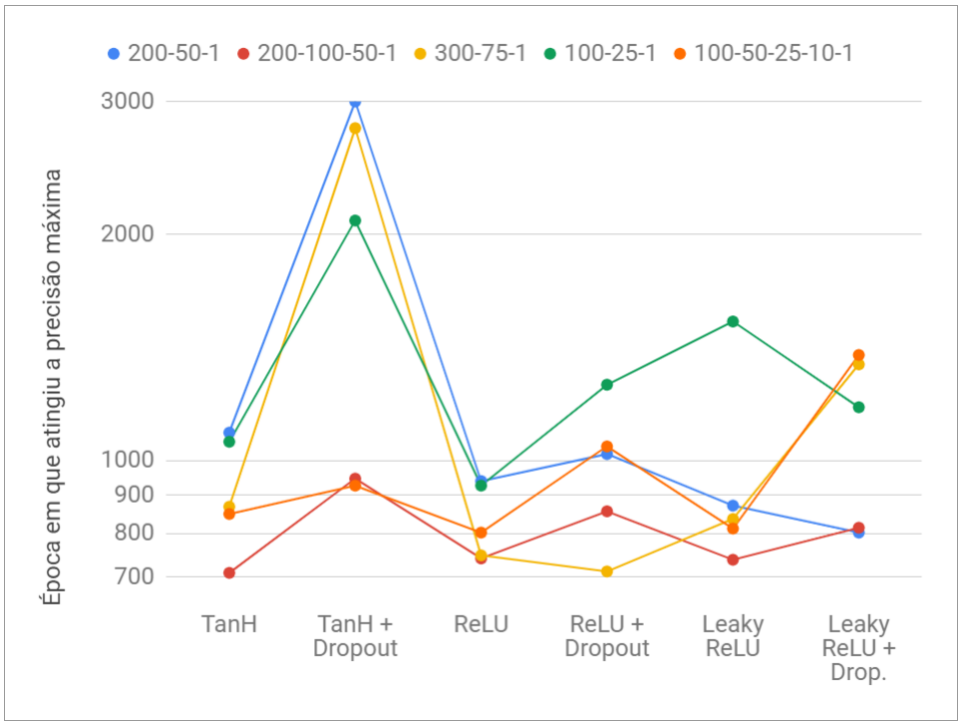
Fonte – Elaborada pelo autor.

Figura 21 – Comparação entre os cenários E, F, G, H, I e J - Precisão máxima alcançada na base de teste, utilizando diferentes configurações de camadas e diferentes funções de ativação.



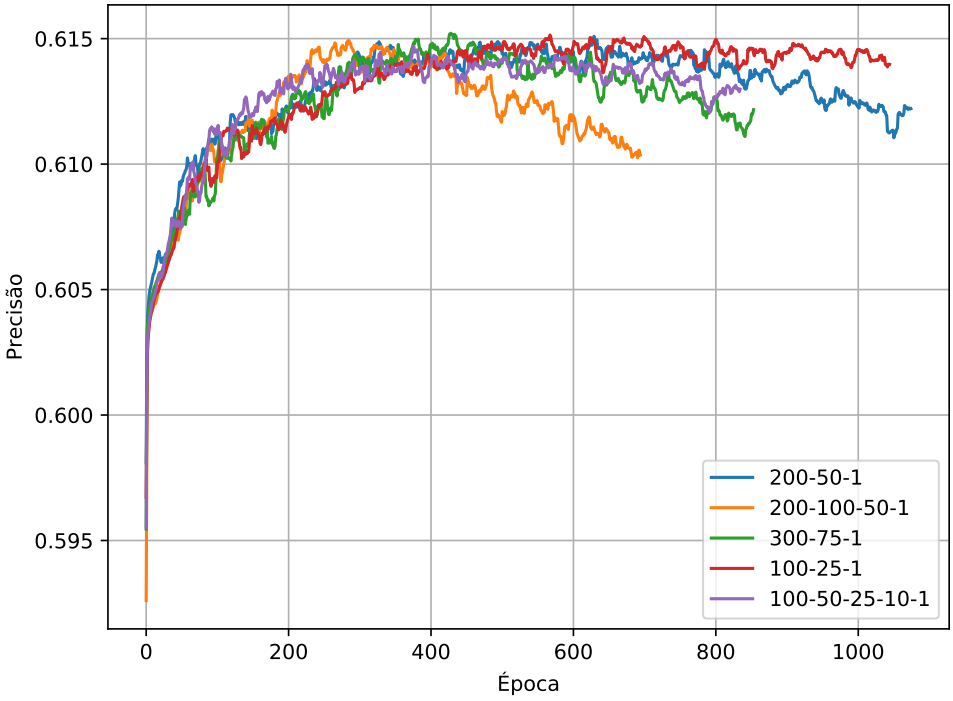
Fonte – Elaborada pelo autor.

Figura 22 – Comparação entre os cenários E, F, G, H, I e J - Época em que foi atingida a precisão máxima na base de teste, utilizando diferentes configurações de camadas e diferentes funções de ativação.



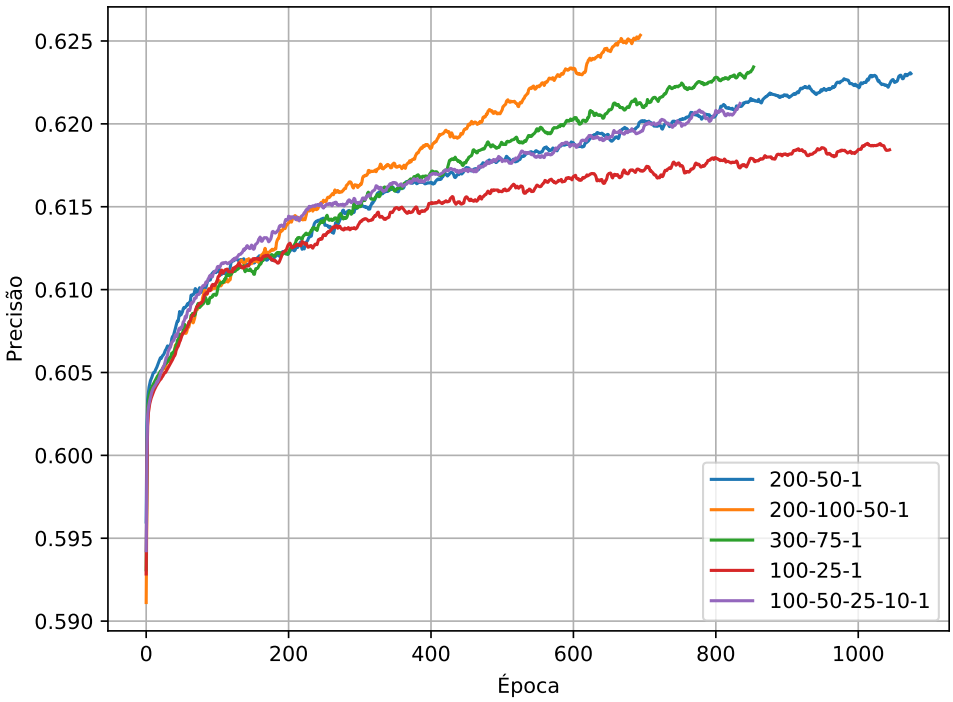
Fonte – Elaborada pelo autor.

Figura 23 – Cenário E - Precisão *versus* época na base de **teste**, utilizando diferentes configurações de camadas. Função de ativação tangente hiperbólica, **sem** utilização de *Dropout*



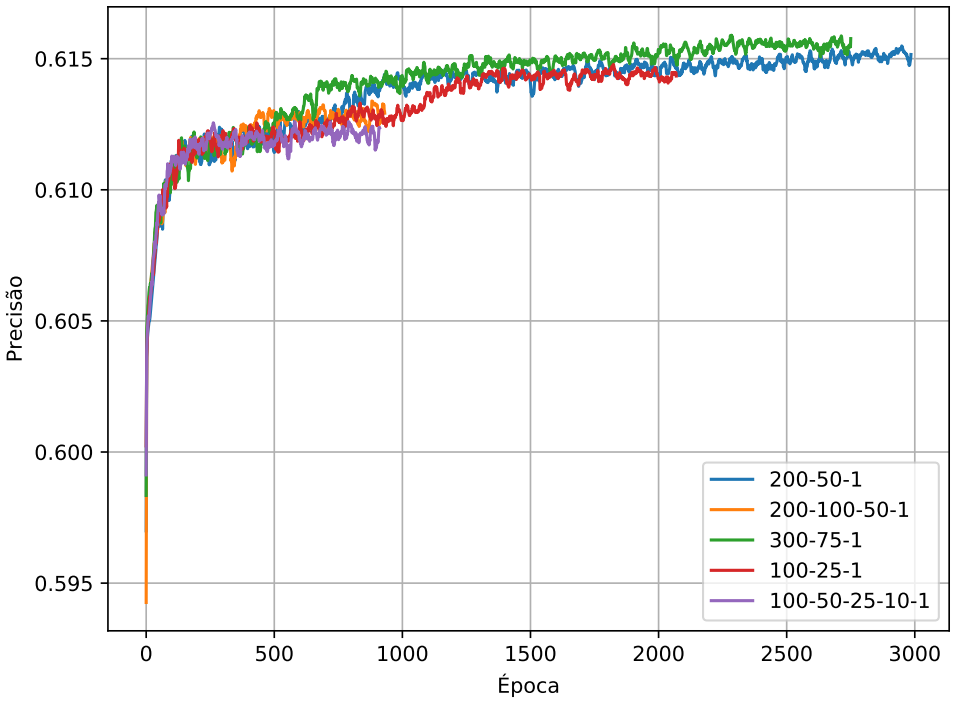
Fonte – Elaborada pelo autor.

Figura 24 – Cenário E - Precisão *versus* época na base de **treino**, utilizando diferentes configurações de camadas. Função de ativação tangente hiperbólica, **sem** utilização de *Dropout*



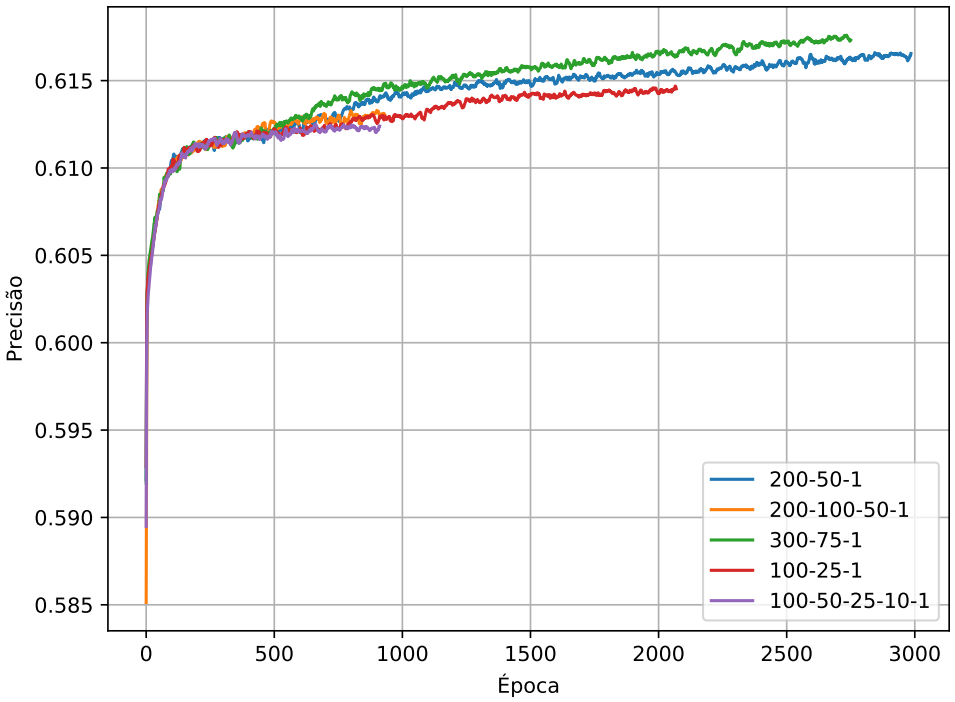
Fonte – Elaborada pelo autor.

Figura 25 – Cenário F - Precisão *versus* época na base de **teste**, em diferentes configurações de camadas. Função de ativação tangente hiperbólica, **com** utilização de *Dropout*



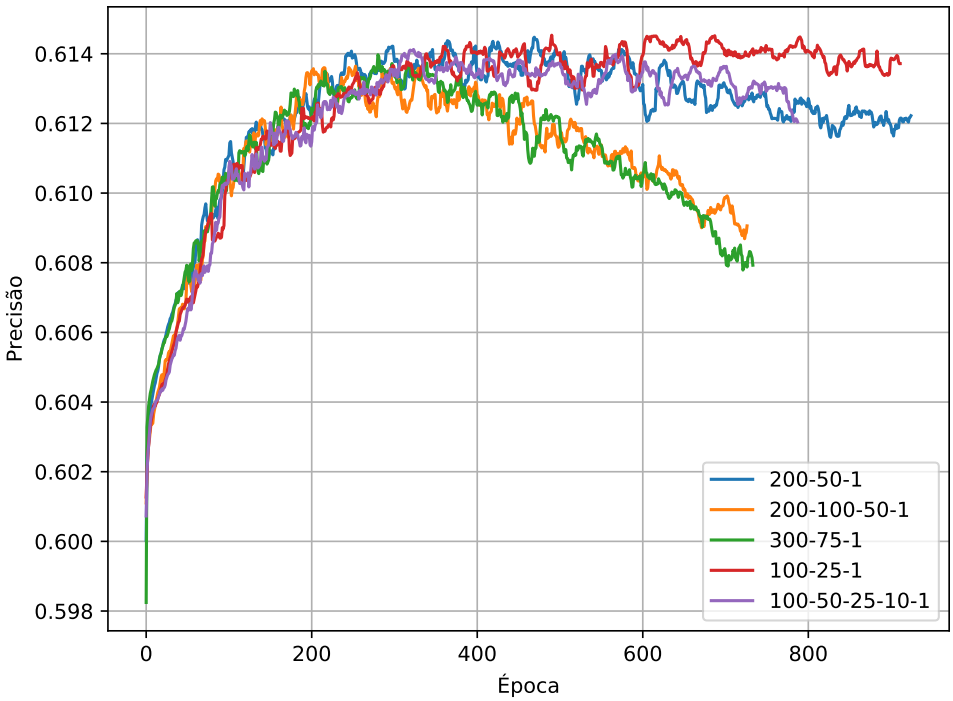
Fonte – Elaborada pelo autor.

Figura 26 – Cenário F - Precisão *versus* época na base de **treino**, em diferentes configurações de camadas. Função de ativação tangente hiperbólica, **com** utilização de *Dropout*



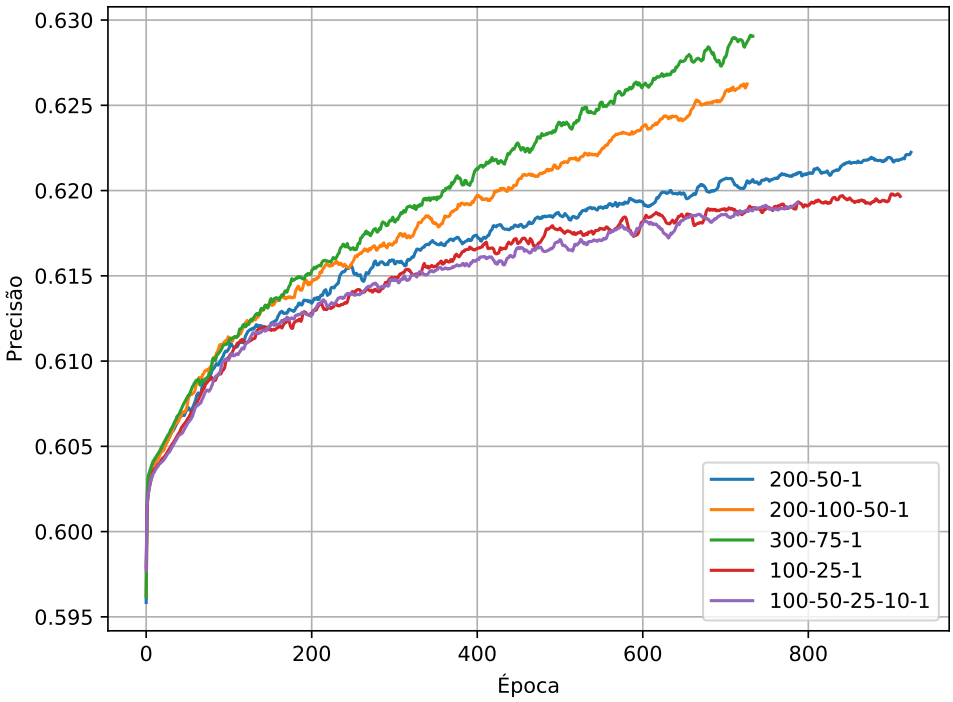
Fonte – Elaborada pelo autor.

Figura 27 – Cenário G - Precisão *versus* época na base de **teste**, utilizando diferentes configurações de camadas. Função de ativação ReLU, **sem** utilização de *Dropout*



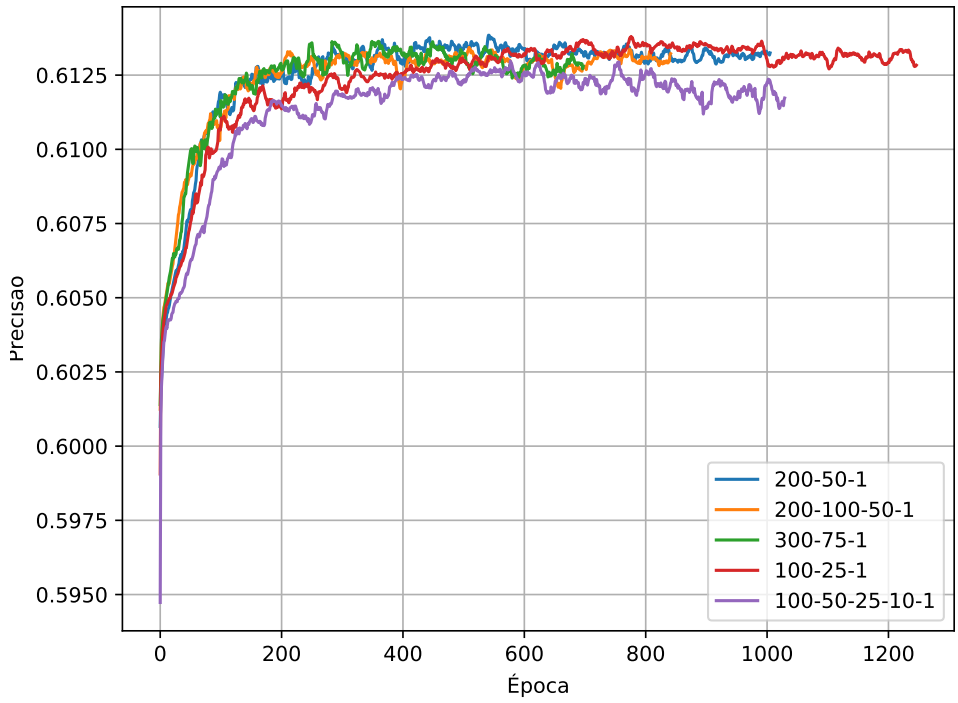
Fonte – Elaborada pelo autor.

Figura 28 – Cenário G - Precisão *versus* época na base de **treino**, utilizando diferentes configurações de camadas. Função de ativação ReLU, **sem** utilização de *Dropout*



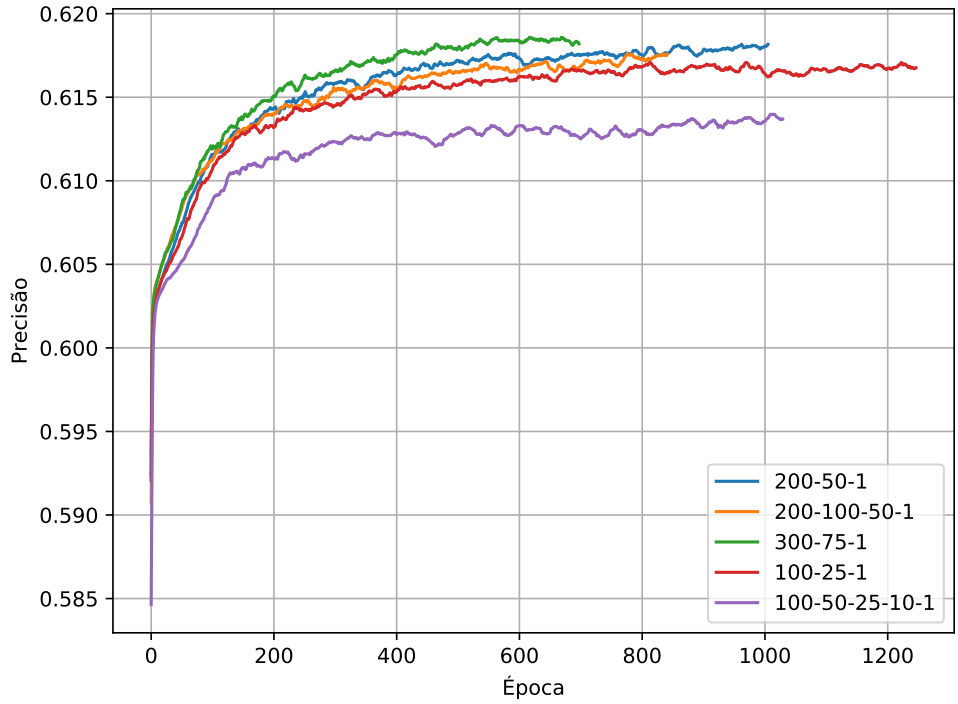
Fonte – Elaborada pelo autor.

Figura 29 – Cenário H - Precisão *versus* época na base de **teste**, utilizando diferentes configurações de camadas. Função de ativação ReLU, **com** utilização de *Dropout*



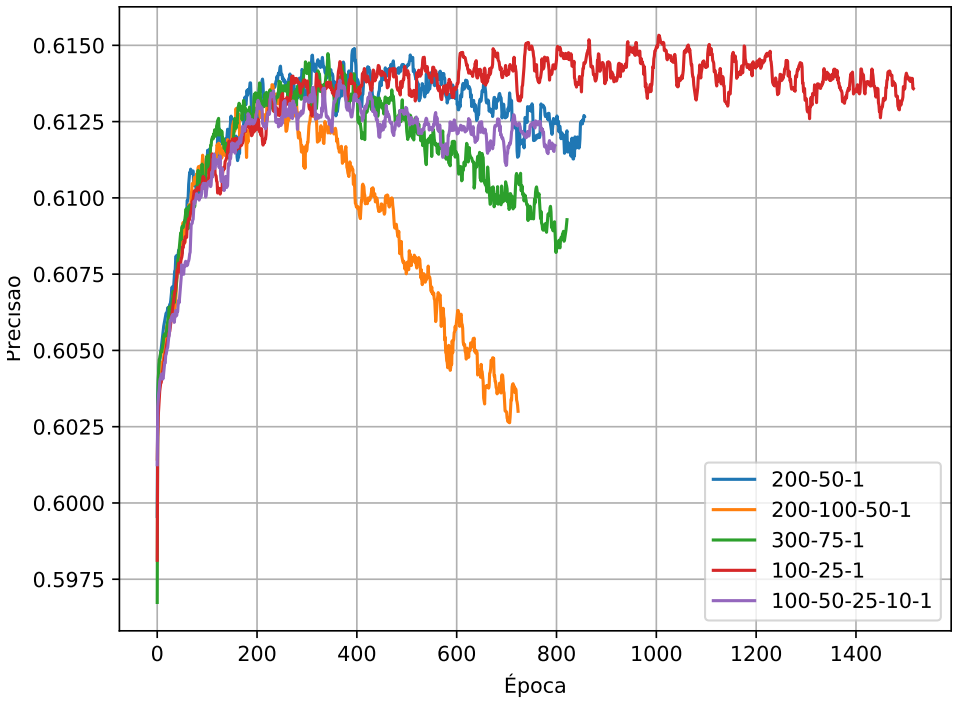
Fonte – Elaborada pelo autor.

Figura 30 – Cenário H - Precisão *versus* época na base de **treino**, utilizando diferentes configurações de camadas. Função de ativação ReLU, **com** utilização de *Dropout*



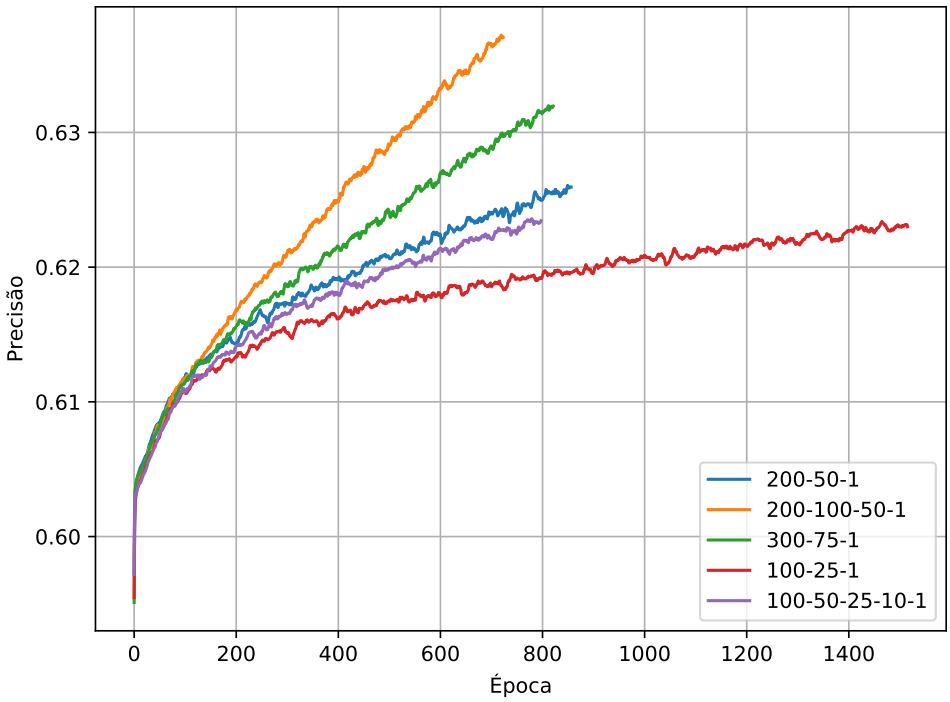
Fonte – Elaborada pelo autor.

Figura 31 – Cenário I - Precisão *versus* época na base de **treino**, utilizando diferentes configurações de camadas. Função de ativação Leaky ReLU, **sem** utilização de *Dropout*



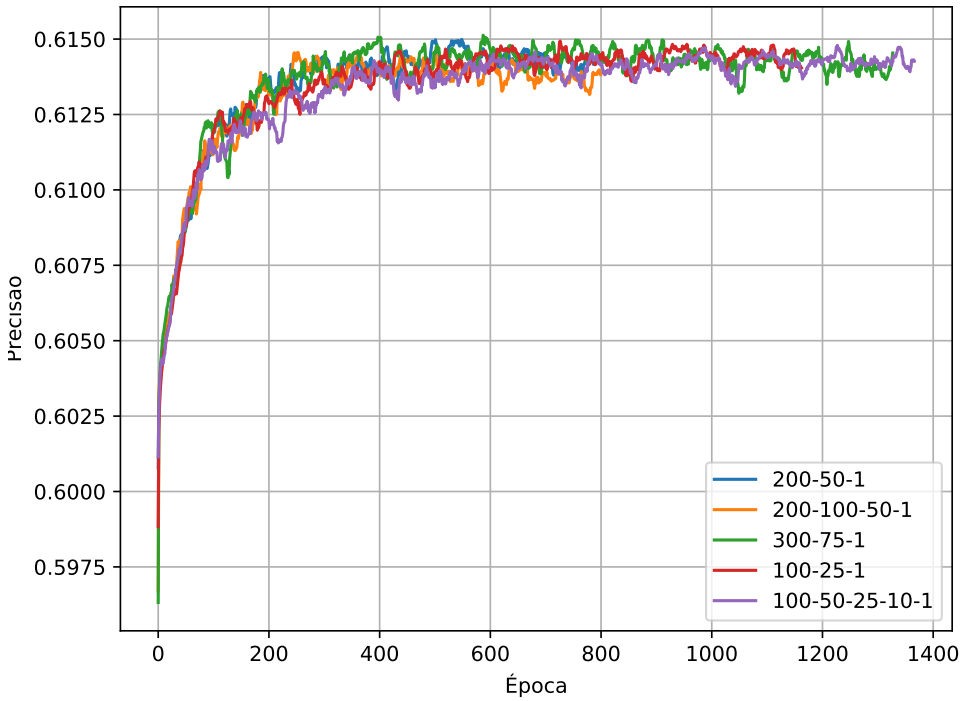
Fonte – Elaborada pelo autor.

Figura 32 – Cenário I - Precisão *versus* época na base de **treino**, utilizando diferentes configurações de camadas. Função de ativação Leaky ReLU, **sem** utilização de *Dropout*



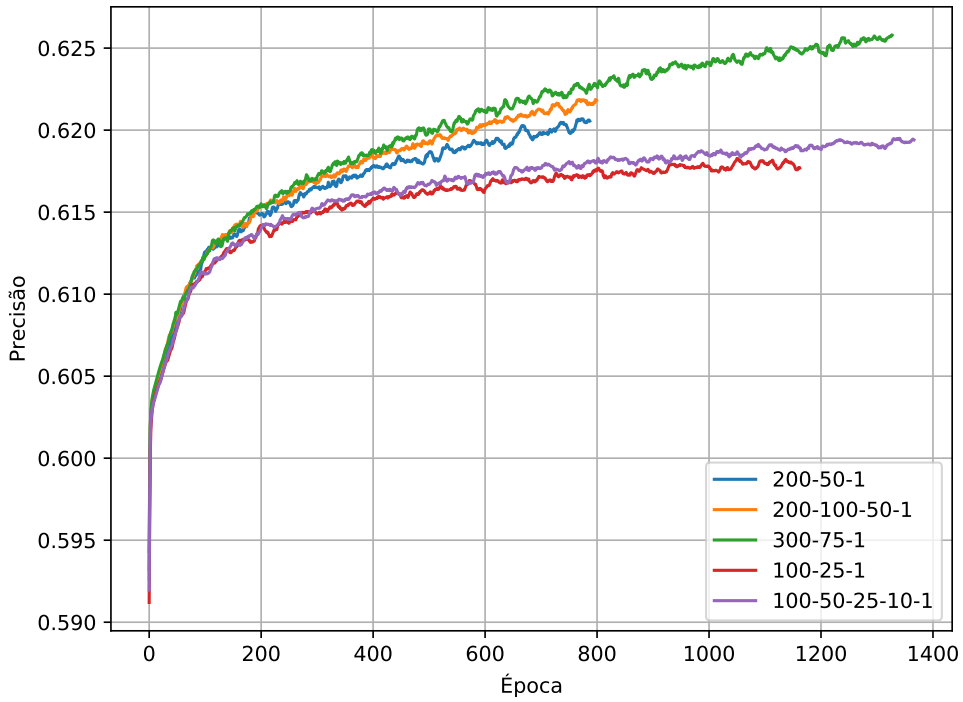
Fonte – Elaborada pelo autor.

Figura 33 – Cenário J - Precisão *versus* época na base de **treino**, utilizando diferentes configurações de camadas. Função de ativação Leaky ReLU, **com** utilização de *Dropout*



Fonte – Elaborada pelo autor.

Figura 34 – Cenário J - Precisão *versus* época na base de **treino**, utilizando diferentes configurações de camadas. Função de ativação Leaky ReLU, **com** utilização de *Dropout*



Fonte – Elaborada pelo autor.

4.1.3 Outros testes

Como citado na seção 4.5, além dos testes destacados acima, também foram realizados inúmeros outros buscando conhecer os efeitos de cada parâmetro. Muitos destes não tiveram seus dados compilados, pois acreditou-se que não seria prático ou útil organizar uma quantidade tão grande de dados.

Neste contexto, esses testes foram utilizados primariamente para aprimorar o conhecimento empírico do autor e guiá-lo para a construção de futuros cenários de teste que fossem mais frutíferos. Abaixo são descritos alguns destes testes.

4.1.3.1 Camadas intermediárias utilizando funções de ativação distintas.

Nos cenários apresentados, quando se variava a função de ativação, utilizava-se esta para todas as camadas intermediárias. No entanto, também foram investigadas configurações em que a primeira camada intermediária utilizava a função ReLU e a camada seguinte utilizava a função tangente hiperbólica, por exemplo.

Foram consideradas várias combinações das funções ReLU, TanH, Sigmoidal e Leaky ReLU, mas determinou-se que não foram relevantes o suficiente no caso geral para o custo adicional, em termos de facilidade de visualização e análise para serem inclusas nos cenários apresentados – apesar de algumas combinações apresentarem resultados interessantes sob certas condições.

4.1.3.2 Velocidade de execução da rede em CPU *vs* GPU

Apesar do *framework* TensorFlow oferecer aceleração em GPU, esta funcionalidade é limitada a modelos específicos de placas da fabricante Nvidia. Dessa forma, embora o ambiente de desenvolvimento deste trabalho possua uma GPU de alto nível, esta é da fabricante AMD e, portanto, o TensorFlow torna-se limitado à execução integral via CPU.

A plataforma PlaidML, por sua vez, permite a execução em GPU nesta placa e o Keras permite cambiar entre TensorFlow e PlaidML utilizando-se uma única linha de configuração, o que conferiu facilidade ao testar os efeitos de execução em CPU vs GPU em diversos cenários. O que pode ser observado nesses testes é que o *hardware* na qual a execução é mais vantajosa varia de acordo com o cenário. Para configurações de rede muito simples ou situações em que o treinamento é realizado utilizando uma quantidade relativamente pequena de amostras, o treinamento é muito mais rápido – até 10 vezes mais – se realizado pela CPU, via TensorFlow. Em contrapartida, caso a configuração da rede seja mais complexa ou, principalmente, caso seja utilizada uma quantidade maior de amostras, a execução em GPU torna-se mais vantajosa, chegando a ser 3 vezes mais rápida.

4.1.3.3 Impactos de diferentes tamanhos de *Batch Size*

Não há dúvida que a escolha de diferentes tamanhos de *batches* para treinamento influenciam não somente a velocidade de treino, mas também a capacidade de convergência. Porém, a literatura apresenta pareceres diferentes sobre o impacto na convergência. Muitos trabalhos, antigos e recentes, defendem que o treinamento em *batches* é quase sempre prejudicial à convergência e que tamanhos maiores de *batches* degradam a capacidade da rede de generalizar seu aprendizado (WILSON; MARTINEZ, 2003) (KESKAR et al., 2016).

Entretanto, concomitantemente, existem trabalhos prestigiosos que indicam o contrário, argumentando que a abordagem em *batches* maiores fornece melhores resultados

de convergência (BRAGA; CARVALHO; LUDERMIR, 2000, p. 66) (SMITH et al., 2017) (GÉRON, 2017, p. 161). A documentação do Keras também recomenda que seja utilizado o treinamento em *batch* e revela ainda que "*(...) é recomendado que se utilize o maior tamanho de batch possível até onde a memória do sistema permitir, já que batches maiores geralmente resultam em uma inferência mais rápida.*"[1] (KERAS, 2018, tradução nossa).

Verificou-se o comportamento da rede utilizando diferentes tamanhos de *batches*, de forma que puderam ser observadas diferenças drásticas em certos cenários em relação à velocidade de processamento, demonstradas nas Tabelas 12 e 13.

A expressiva diferença de comportamento entre processamento por GPU e CPU, nesses casos, pode ser justificada pelo gargalo que a GPU sofre para transferir os dados da memória principal para sua própria memória. Em *batches* pequenas, o atraso causado por este tipo de operação se torna o fator limitante de desempenho, enquanto em *batches* maiores, ela compensa este atraso com sua velocidade de processamento superior.

Tabela 12 – Tempo gasto por época de treinamento, utilizando 800.000 amostras, processando em **GPU**, via PlaidML, numa rede com topologia de camadas intermediárias 200-100-50

Tamanho do *Batch*	Tempo gasto por época (hh:mm:ss)
1	1:05:12
8	0:09:15
80	0:00:58
800	0:00:08
8000	0:00:02
80.000	0:00:02

Tabela 13 – Tempo gasto por época de treinamento, utilizando 800.000 amostras, processando em **CPU**, via TensorFlow, numa rede com topologia de camadas intermediárias 200-100-50

Tamanho do *Batch*	Tempo gasto por época (hh:mm:ss)
1	0:08:05
8	0:01:01
80	0:00:15
800	0:00:06
8000	0:00:05
80.000	0:00:05

[1] "*(...) it is recommended to pick a batch size that is as large as you can afford without going out of memory (since larger batches will usually result in faster evaluation/prediction).*"

Quanto à relação com a capacidade de convergência da rede, os resultados encontrados também suportam a noção de que tamanhos maiores de *batches* são mais eficazes. Para exemplificar isto, construiu-se a a Tabela 14, realizando 60 treinos nas mesmas configurações, variando-se apenas o tamanho do *batch* – que foi desde 5% da quantidade de amostras utilizadas no treino até 100%. O restante das configurações foram as seguintes:

- 10.000 amostras, 8.000 para treino e 2.000 para teste;
- Rede com topologia de camadas intermediárias 100-25;
- *Dropout* de 0,2 e 0,05 nas camadas intermediárias;
- Otimizador Adam;
- Função de ativação tangente hiperbólica;
- Processamento em CPU, via TensorFlow, visto que é mais rápido para bases menores.

Embora os dois treinos com *batches* de maior tamanho tenham levado um tempo ligeiramente maior para convergir, eles convergiram em todos os treinos; enquanto isso, os outros treinos com tamanhos menores de *batches* não. Em metade das vezes, os treinos com *batch* = 2500 alcançaram uma precisão máxima de apenas 64,99%, enquanto os treinos com *batch* = 1000 e *batch* = 500 tiveram precisão máxima de 64,89% e 64,74%, respectivamente. São diferenças relativamente pequenas, mas que se alinham com a parcela da literatura que defende que tamanhos de *batches* maiores propiciam uma melhor convergência.

Tabela 14 – Tempo gasto para atingir 65% de precisão na base de teste utilizando diferentes tamanhos de *batches*

Tamanho do Batch	10.000	5000	2500	1000	500
Tempo necessário para convergir (segundos)	24.5	23.3	22.0	–	–
	23.8	22.7	22.0	–	–
	22.1	22.4	–	–	–
	23.9	22.6	–	–	–
	26.7	26.3	–	–	–
	24.1	23.1	21.7	–	–
	23.6	20.6	21.0	–	–
	22.1	22.4	–	–	–
	27.4	22.8	21.8	–	–
	23.6	22.4	–	–	–
	24.0	22.5	–	–	–
	23.8	25.5	22.5	–	–

Fonte – Elaborada pelo autor.

4.1.3.4 Topologias extremas

Um outro tipo de topologia que foi investigado é o das redes muito complexas. Foram treinados modelos utilizando números significativamente maiores de neurônios por camada ou de quantidade de camadas. A configuração mais profunda chegou a utilizar 20 camadas, porém não pôde ser observada nenhuma melhora significativa no modelo, tanto em precisão máxima atingida, quanto em velocidade de convergência.

As principais diferenças foram um aumento no tempo de treino por época e uma capacidade maior de realizar *overfitting*, especialmente caso não seja utilizado *Dropout*. Por esses motivos, não serão apresentados gráficos e tabelas acerca desses cenários, pois não acrescentariam informações valiosas.

4.1.3.5 Treinos sem condições de Parada Antecipada

Também foi averiguado o comportamento da rede caso fossem removidas as condições de parada que almejam impedir o *overfitting* e fosse permitido à rede que permanecesse treinando por extensos períodos. Foram realizados acima de 10 tentativas nessa modalidade, em diferentes configurações. Abaixo é descrito um dos cenários averiguados.

- 100.000 amostras, 80.000 para treino e 20.000 para teste;
- Rede com topologia de camadas intermediárias 200-100;
- Sem *Dropout*;
- Otimizador Adam;
- Função de ativação tangente hiperbólica;
- Processamento em GPU, via PlaidML.

No decorrer de 16 horas e 11 minutos, a rede realizou 99.999 épocas de treino. Apesar do longo período, a rede apresentou péssimos resultados e características claras de *overfitting* – a precisão na base de treino era próxima de 81,5%, enquanto a precisão na base de teste encontrava-se pouco acima de 55%. Testes em outros cenários revelaram que aumentar o número de camadas permite um grau ainda maior de *overfitting*, sendo capazes de chegar a 98% de precisão na base de treino e próximos a 50% de precisão na base de teste.

4.1.4 Configuração final proposta

Tendo como base os testes realizados, foi proposta uma topologia de rede para atender ao problema, a ser treinada com as seguintes configurações:

- Quantidade de entradas: 250 entradas, descritas em detalhes na seção 3.2.4;
- Quantidade de camadas intermediárias: 3;
  - Quantidade de neurônios na 1ª camada intermediária: 200;
  - Quantidade de neurônios na 2ª camada intermediária: 100;
  - Quantidade de neurônios na 3ª camada intermediária: 50;
  - Função de ativação em todas as camadas intermediárias: tangente hiperbólica.

- Quantidade de saídas: Uma única saída, de 0 a 1, de tal forma que 0 representa a vitória do time *Dire* e 1 representa a vitória do time *Radiant*;
- Otimizador: Adam;
- Tamanho da base de dados: 800.000 registros de partidas;
- Quantidade de amostras utilizadas para treino: 80% - 640.000;
- Quantidade de amostras utilizadas para teste: 20% - 160.000;
- Utilização da técnica de *Dropout*: Não utilizada;
- Quantidade máxima de épocas de treino: 5.000;
- Condições de parada antecipada:
  - Caso a taxa de acertos na base de treino atinja 75%;
  - Caso a taxa de acertos na base de teste permaneça durante 464 épocas consecutivas sem apresentar uma melhora de ao menos 0.00001.
- Tamanho do *batch*: 640.000;
- Plataforma de treino: GPU, via PlaidML.

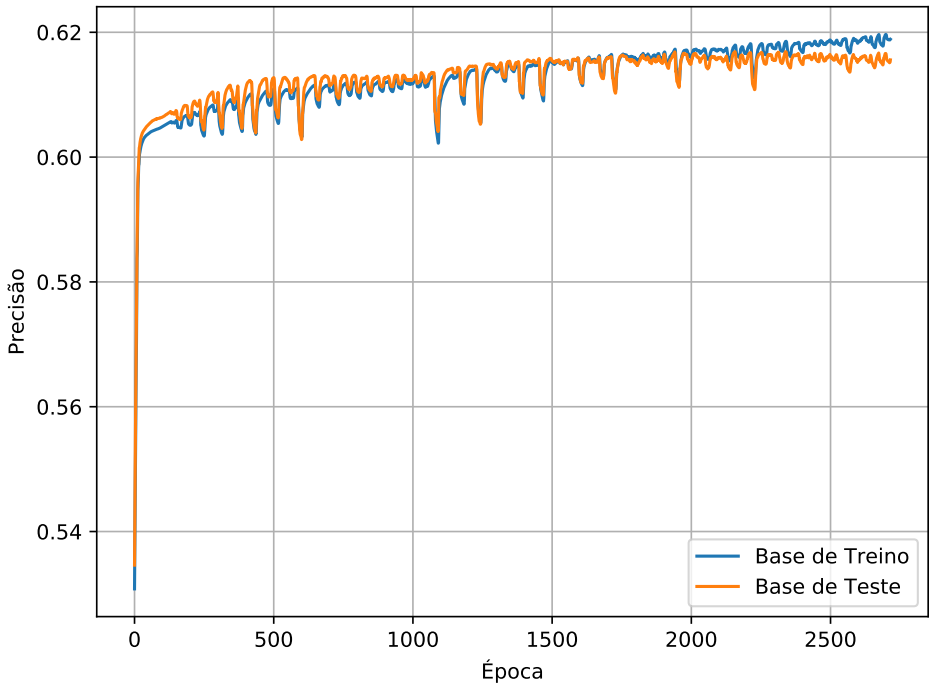
A rede realizou 2.730 épocas de treinamento, até ser interrompida antecipadamente em função da estagnação da taxa de acertos na base de teste. Cada época de treino foi realizada em aproximadamente 2,3 segundos e o treino completo da rede precisou de 1 hora, 48 minutos e 50 segundos para ser concluído.

Foi alcançada uma precisão de 61,78% na base de teste e de 62,01% na base de treino. A Figura 35 mostra o comportamento da rede durante o treinamento. A melhor precisão na base de teste foi atingida na época de número 2.265.

Afim de verificar a robustez do modelo em relação à sua capacidade de generalização, foi realizado também um treino num cenário extremo, no qual a divisão das amostras em base de treino e teste foi feita de forma a fornecer ao sistema apenas 10% das amostras para treino, no lugar dos 80% usuais. Dessa maneira, o sistema utilizou apenas 80.000 amostras para treino e tinha suas previsões de vitória conferidas em 720.000 partidas, as quais ele nunca teve oportunidade de processar. O restante das configurações foram mantidas idênticas às anteriores.

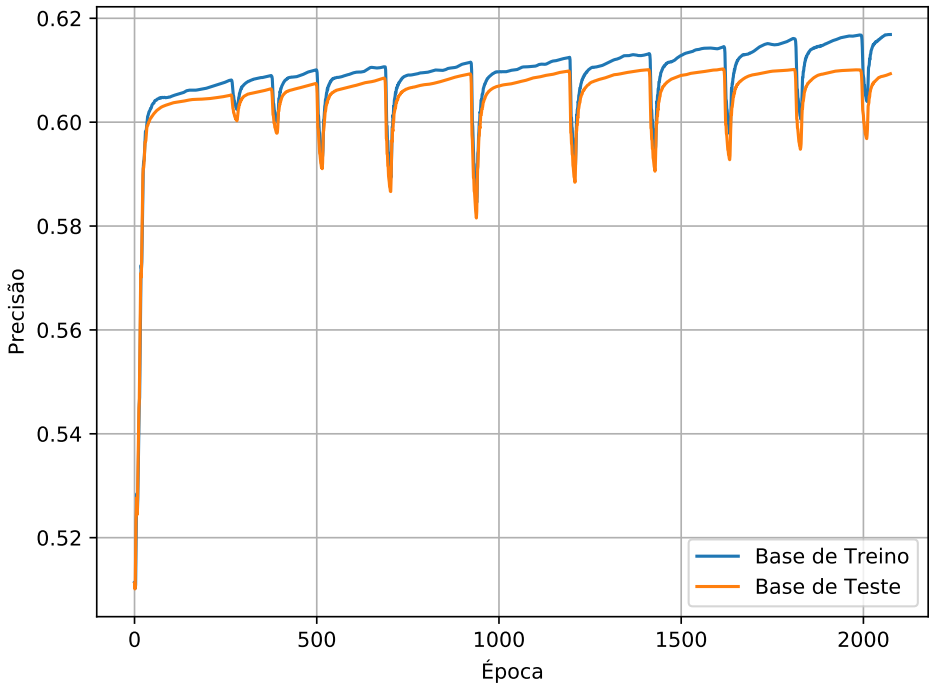
Mesmo nesse cenário extremo, a rede conseguiu atingir precisão de 61,04% na base de teste e 61,73% na base de treino, demonstrando que o modelo apresenta boa capacidade de generalização. A Figura 36 apresenta o comportamento durante o treino.

Figura 35 – Precisão *versus* época para a topologia de rede proposta



Fonte – Elaborada pelo autor.

Figura 36 – Precisão *versus* época para a topologia de rede proposta, num cenário extremo, utilizando apenas 80.000 amostras para treino e 720.000 para teste



Fonte – Elaborada pelo autor.

5 Discussões

“É fácil ser sábio depois que já aconteceu.”
Arthur Conan Doyle

Embora o percentual de acerto alcançado seja suficientemente expressivo para permitir que a rede seja utilizada como plataforma para ferramentas que necessitem de estimativas de vitória, os resultados acima ficaram abaixo das expectativas *a priori* deste autor, pois estas encontravam-se previamente influenciadas por outros trabalhos.

Por exemplo, o trabalho realizado por (CONLEY; PERRY, 2013), propõe-se a resolver o mesmo problema aqui investigado, porém utilizando outra abordagem: um algorítimo KNN para realizar regressão logística. Com isso, e utilizando uma base de dados composta por 56.691 partidas coletadas durante 33 dias, os autores obtiveram 69,8% de precisão em prever qual time seria vitorioso para a base de teste, baseando-se apenas nos heróis escolhidos.

A seguir, foram construídas hipóteses que acredita-se justificar parcialmente esta diferença entre ambos resultados e, neste processo, também percebeu-se não ser possível uma comparação direta entre eles. Ultimamente, é argumentado ainda que, sob condições mais adequadas, o modelo proposto poderia fornecer resultados melhores.

5.0.1 Diferença entre resultados

Apesar de ambos os trabalhos se proporem a resolver o mesmo problema, comparações diretas entre o resultado de ambos não se fazem válidas, pois eles utilizam bases amostrais vastamente diferentes. Por exemplo, a base aqui utilizada é maior que a outra em mais de uma ordem de magnitude, possuindo mais de 14 vezes mais registros. Para exemplificar os efeitos possíveis desta diferença e corroborar a inviabilidade de comparações, é possível relatar uma situação muito interessante ocorrida durante a realização dos treinos no desenvolvimento deste trabalho.

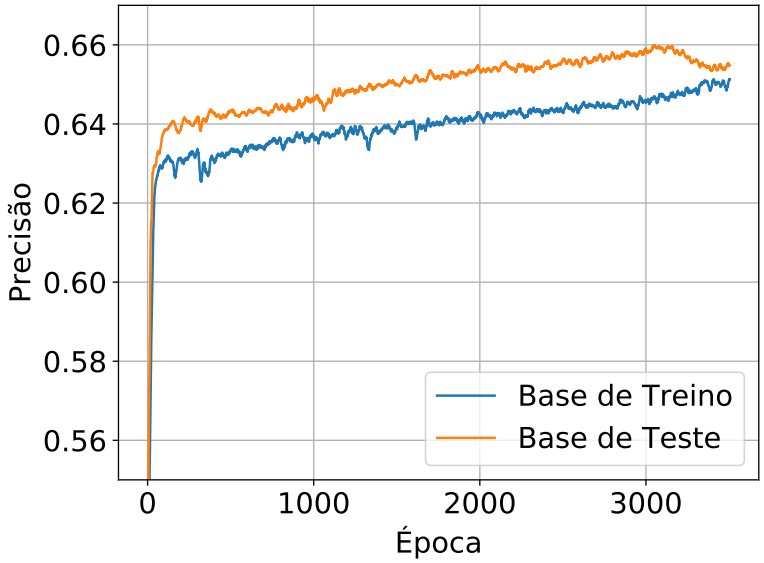
Foi observado que, utilizando tamanhos menores de base amostral, é possível obter uma taxa de acertos de previsões substancialmente maior do que com a utilização da base completa com as 800.000 partidas. Por exemplo, em um treino utilizando uma parcela de apenas 10.000 amostras, foi possível obter uma precisão de 66,3% de acerto na base de teste – uma diferença de 5% em relação à precisão máxima alcançada utilizando a base completa.

A Figura 37 apresenta um desses treinos. Pode-se destacar ainda que, devido à pequena quantidade de amostras, o tempo necessário para convergência da rede foi de apenas 1 minuto e 41 segundos.

Com base nessas observações, pode-se inferir que, mesmo exercendo certas cautelas durante a coleta e filtragem de amostras, a base de dados adquirida pode possuir diferentes distribuições e agrupamentos de dados.

Em retrospectiva, isso não é difícil de se justificar, pois as informações presentes em uma base dependem diretamente do desempenho de jogadores humanos com determinados heróis em determinada época, sendo que humanos naturalmente estão em constante aprendizado e mudança na forma como operam.

Figura 37 – Precisão *versus* época. Treino utilizando 10.000 amostras, dividias em 8.000 para treino e 2.000 para teste. Função de ativação TanH, otimizador Adam, topologia de camadas intermediárias 100-25, utilizando *Dropout*



Fonte – Elaborada pelo autor.

Isoladamente, esta constante mudança humana já é capaz de criar situações em que, no decorrer de suas experiências no jogo, um mesmo jogador pode apresentar um certo desempenho com um herói em certa época do ano e, posteriormente, um desempenho diferente. Porém, além disso, existe um conceito frequentemente utilizado pelas comunidades de jogos eletrônicos competitivos, que acredita-se ser capaz de auxiliar a justificativa da variância na distribuição de amostras – o chamado *metagame*.

5.0.2 *Metagame*

O antepositivo *meta* advém da língua Grega e significa "além". O termo foi adotado por diversas línguas como indicação de algo que transcende ou que vai além de outra coisa. No contexto de jogos, o *metagame* pode ser traduzido como "o jogo além do jogo"e consiste em qualquer abordagem que transcenda ou opere fora das regras pré-determinadas do jogo ou utilize fatores externos para afetar o jogo (HOWARD, 1971) (CERRATO, 2016).

De maneira superficial, pode ser descrito em ações ou comportamentos dos jogadores de um jogo, que são motivados não pelas regras que definem o jogo, mas sim pelas ações e comportamentos de outros jogadores. Para melhor entendimento, pode-se citar exemplos em esportes tradicionais:

- Em xadrez, um competidor A observa diversos jogos de um competidor B e que nos últimos 5 jogos consecutivos o jogador B tentou executar uma determinada estratégia muito específica no xadrez. Se posteriormente o competidor A enfrentar o competidor B numa partida e jogar premeditadamente de uma maneira que dê a ele uma vantagem, caso o competidor B repita aquela estratégia, poderá ser dito que o jogador A tirou proveito do *metagame*;

- Em futebol, grandes times estudam seus adversários previamente, analisando partidas passadas para identificar suas estratégias preferidas, tendências e fraquezas. Dessa maneira, podem formular uma estratégia de jogo específica contra aquele adversário que lhes forneça mais chances de vitória. Isso também é chamado de *metagame*.

Em ambos os casos descritos, as ações dos jogadores foram guiadas por fatores externos e não pelas regras e configurações dos jogos em si. Em jogos eletrônicos, este fenômeno se faz muito presente e ainda mais impactante, geralmente afetando toda a comunidade. Como um exemplo em DOTA 2, pode-se citar a seguinte situação, que não é incomum:

1. Um jogador adota uma estratégia não usual para um certo herói e obtém resultados excepcionais na partida;
2. Os outros 9 jogadores presentes na partida tomam nota e eventualmente também adotam a estratégia para aquele herói em outras partidas, também obtendo bons resultados;
3. Num período de tempo relativamente curto, é possível que grande parte da comunidade daquele jogo também esteja adotando aquela estratégia – que, por sua vez, passa a ser cada vez menos eficaz à medida que vai se tornando mais conhecida, pois os outros jogadores começam a se adaptar e tomar decisões que limitem a efetividade daquela estratégia.

Outro exemplo corriqueiro é a adoção massiva de determinado herói pela comunidade após um jogador profissional realizar um feito incomum em algum grande evento ou campeonato. Os resultados obtidos pelo jogador não precisam ser bons em termos de auxiliar a vitória de seu time – causar um efeito cômico e divertido já é suficiente para influenciar o resto dos jogadores a tentar fazer o mesmo. Da mesma forma, em pouco tempo a comunidade desenvolve medidas para anular tal jogada.

Diante disso, é plausível supor que, além das flutuações naturais no nível de habilidade dos jogadores, esse efeito agrave ainda mais a variabilidade das amostras. Por exemplo, considerando-se um jogador que tenha bons resultados com um determinado herói, para que seu desempenho mude significativamente não é necessário que aconteça uma mudança de versão do jogo e nem que suas habilidades melhorem ou piorem – basta que ele fique poucas semanas sem jogar e o estado do *metagame* mude o suficiente.

Um reflexo claro disso que pode ser visualizado em DOTA 2 é o fato das taxas de vitória médias de muitos heróis mudarem de forma frequente e considerável, mesmo na ausência de qualquer *patch* de balanceamento ou mudança nas regras de jogo (STRATZ, 2018).

Tendo em vista que o modelo proposto utiliza como parte fundamental de suas entradas as taxas médias de vitória dos heróis, não é surpreendente que a utilização de uma base de dados maior para estas inferências seja prejudicial ao desempenho do modelo. Coletar amostras durante um intervalo maior de tempo resultaria em taxas médias de vitória artificiais, que na prática não correspondem a nenhuma época específica e, sim, a todo o período.

Por esses motivos, considera-se razoável supor que para jogos muito suscetíveis a este fenômeno, uma base de dados se torna progressivamente menos consistente à medida em que se aumenta o período sobre o qual são coletadas as amostras.

Se isso for verdade, também seria coerente com o menor desempenho, já que a base aqui utilizada foi coletada durante um período de tempo aproximadamente quatro vezes maior que a base utilizada por (CONLEY; PERRY, 2013). Além disso, mesmo nos cenários em que foram utilizadas parcelas menores da base, os dados utilizados para representar cada herói ainda foram advindos da base integral, pois foram inferidos previamente e apenas uma vez, utilizando toda a base.

Em contrapartida, isso também significaria que seria possível aumentar ainda mais o desempenho da rede, a partir de uma simples inferência de dados utilizando quantidades menores de amostras.

5.0.3 Precisão máxima possível de estimativas

Embora o raciocínio apresentado acima argumente que se acredita ser possível obter uma melhora de precisão no modelo, existe ainda uma outra hipótese digna de ser mencionada. Esta se faz relevante, pois caso verdadeira, os resultados poderiam ser interpretados de maneira ainda mais otimista.

Para fundamentar a hipótese, considera-se útil relembrar e detalhar o ponto citado na seção 1.2 durante a descrição do problema, onde afirma-se que faz parte do conhecimento empírico dos jogadores o fato de que a escolha dos heróis em cada time pode ser utilizada para se estimar qual será o time vencedor (GODEC, 2015).

Como exemplo disso, pode ser citado o jogador Peter Dager (LIQUIPEDIA, 2014). Campeão do *The International* em 2015 e considerado um dos melhores jogadores do mundo (BONIFACIO, 2018), é reconhecido como um ótimo capitão de time, principalmente por sua notória habilidade de usualmente conseguir selecionar heróis de maneira superior aos capitães adversários no começo de uma partida (MURESAN, 2018).

De fato, especialmente se tratando de partidas com jogadores de alto nível de habilidade, entende-se que *"(...) a fase de seleção de heróis é essencialmente uma partida de xadrez entre os capitães dos dois times, que por vezes pode decidir o resultado da partida antes mesmo dela começar."*[1] (BONIFACIO, 2017, tradução nossa).

Partindo dessa premissa, pode-se inferir que em algumas partidas é possível estimar corretamente as chances de vitória de um time baseando-se apenas nos heróis selecionados, com taxa de acerto superior a 50%.

Em contrapartida, o jogo é jogado por jogadores humanos e ultimamente é o comportamento humano que decide o resultado individual de cada partida. Um simples lapso momentâneo de atenção ou um erro realizado por um único jogador em menos de um segundo podem ser suficientes para mudar completamente os rumos de uma partida, em que outrora a vitória já estaria praticamente determinada.

Existem incontáveis exemplos disso e para melhor entendimento podemos citar um ocorrido no *The International* de 2013, que ficou conhecido como "O Erro de 345.000 mil dólares"[2](JACKSON, 2013, tradução nossa). Aos 32 minutos de partida, um jogador do time que até então estava ganhando posicionou seu cursor um centímetro errado e acabou acidentalmente destruindo um item que auxiliaria bastante rumo à vitória e, por fim, seu time acabou perdendo a partida.

[1] *"(...) drafting phase is essentially a chess match between the two team captains (or whoever is drafting for the team if they have a different setup) which can at times decide the outcome of the game before it even begins."*

[2] *The $345,000 Mistake*

Com base nessa outra premissa, pode-se entender que é impossível prever com taxa de acerto de 100% as chances de vitória de um time. Caso esses dois fatos sejam considerados verdadeiros, é possível argumentar que, apesar de ser possível prever a vitória baseando-se apenas em personagens, a taxa de acerto máxima possível pode estar em qualquer ponto superior a 50% e inferior a 100% e não parece plausível que esta seja determinada analiticamente, devido à incerteza e abstração introduzida pela presença de jogadores humanos.

Esta impossibilidade de identificar o máximo de acerto que seria possível alcançar implicaria que, para realmente avaliar o desempenho, seria necessário mensurar os resultados fornecidos por outros modelos, utilizando a mesma base de dados para todos. Qualquer outra tentativa de avaliação se tornaria menos interessante, pois não seria implausível que a rede tenha alcançado uma taxa de acerto apenas marginalmente abaixo da máxima que seria possível para a base utilizada, porém não há como conferir isto.

6 Considerações finais

“Não me venham com conclusões!
A única conclusão é morrer.”
Fernando Pessoa

O mercado mundial de jogos eletrônicos movimenta centenas de bilhões de dólares por ano, sendo que está em constante crescimento. Durante uma partida do jogo DOTA 2, a escolha dos heróis possui expressiva importância, podendo influenciar o resultado antes mesmo do jogo começar. Embora esse fato seja reconhecido pelos jogadores, saber qual personagem escolher não é uma tarefa fácil.

Em vista disso, este trabalho se propôs a desenvolver um modelo que pudesse aprender a estimar qual seria o provável time vencedor de uma partida, baseando-se apenas nos heróis escolhidos em cada time. Um modelo com tais capacidades poderia ser utilizado como base para um sistema de recomendação de escolha de heróis ou em apostas sobre os resultados dos jogos.

Como método para alcançar esse objetivo, foi realizado um levantamento acerca de trabalhos com objetivo similar e seus princípios de funcionamento. Optou-se por construir uma rede neural artificial do tipo MLP, que utilizaria registros de 800.000 partidas reais como base de dados para treinamento.

O modelo de rede proposto foi capaz de acertar o time vitorioso 61,7% das vezes e apresentou boa capacidade de generalização, conseguindo também estimar o resultado de uma quantidade de partidas muito maior que a utilizada durante a aprendizagem. Além disso, conseguiu-se treinar uma rede similar que obteve 66,3% de acerto quando limitada a um subconjunto dos registros de partidas.

Embora existam outros trabalhos com o mesmo problema que tenham apresentado uma maior precisão, os resultados não podem ser diretamente comparados, pois cada trabalho utilizou bases de amostras diferentes e a evidência sugere que diferentes bases de dados podem ter diferentes limites de estimativas de vitória.

Em especial, há razões para crer que, mesmo limitando-se a uma mesma versão do jogo, quanto maior for o o intervalo de tempo em que são coletadas as amostras, menor será a precisão máxima de acertos que poderá ser atingida utilizando-se aquela base.

Finalmente, acredita-se haver possibilidade de alcançar melhoras significativas no desempenho do modelo desenvolvido, caso sejam realizadas certas mudanças no processo de construção da base de dados que será utilizada.

6.1 Limitações

Em determinado momento no decorrer do desenvolvimento do projeto, foram observados alguns resultados que indicariam ser possível uma melhora, caso fossem realizadas mudanças na base de dados utilizada. Contudo, por restrições de tempo, não foi possível averiguar essa hipótese em tempo hábil para a data estipulada de conclusão do projeto.

Apesar de existirem na literatura de redes neurais artificiais diversos métodos propostos para auxiliar no processo de ajustes finos das mesmas, neste trabalho não foram

utilizados tais métodos, pois acreditou-se que poderiam limitar o entendimento do autor sobre o modelo, visto que muitos destes métodos realizam ajustes de forma opaca, como uma ”caixa preta”.

6.2 Trabalhos Futuros

A partir deste trabalho, é possível especular certas linhas de pesquisa pertinentes para serem expandidas, pois possivelmente trariam resultados relevantes para o problema. Dentre essas, destacam-se:

6.2.1 Investigar importância das bases de dados

Acredita-se que as características das bases de dados podem influenciar profundamente a qualidade dos resultados que podem ser obtidos. Com isso em vista, sugere-se que sejam investigados os efeitos nos resultados ao se utilizar bases diferentes, variando, por exemplo:

- Intervalo de tempo sobre o qual é realizada a coleta de amostras;
- Segregação entre diferentes modos de jogo;
- Utilização de partidas normais ou ranqueadas;
- Inclusão de partidas de diferentes níveis de habilidade de jogadores.

6.2.2 Implementação KNM

O trabalho descrito por (CONLEY; PERRY, 2013) sugere ser possível utilizar as escolhas dos heróis em cada time para alcançar aproximadamente 70% de precisão em estimação de vitória. Seria interessante realizar uma comparação entre ambos os métodos, porém, como os autores criaram sua própria base e não a disponibilizaram, não seria adequado fazer uma comparação direta.

Sendo assim, é sugerido replicar o algoritmo por eles proposto e comparar os resultados utilizando uma mesma base, pois acredita-se que esta comparação seria frutífera e poderia facilitar uma melhor compreensão das características do problema.

6.2.3 *Xavier initialization*

Um grande problema presente no desenvolvimento de redes neurais é a imprevisibilidade da convergência do treinamento, que pode variar significativamente (WANG, 1995). Os valores iniciais escolhidos para os pesos podem determinar não apenas o tempo de convergência, mas também a convergência em si, sendo possível que a inicialização adequada de pesos leve o algoritmo a convergir, enquanto uma inicialização ruim leve-o à estagnação em um mínimo local (JOSHI, 2016).

Para inicialização de pesos, este trabalho utilizou o processo mais comum, que consiste em inicializar pesos aleatórios. O método proposto por (GLOROT; BENGIO, 2010), conhecido como *Xavier initialization*, se propõe a aprimorar esta situação e, desde sua publicação em 2010, vem sendo adotado com bons resultados (GEORGE, 2017).

6.2.4 *One-hot input encoding*

Uma outra maneira de lidar com o problema de entradas que representam categorias abstratas é a codificação *One-hot* (BROWNLEE, 2017) (LEARNING, 2018). Embora o uso deste tipo de codificação ampliaria significativamente o número de entradas necessárias, ela transmitiria menos informações pré-processadas para a rede, fornecendo os dados da forma mais direta e transparente possível.

Dessa maneira, apesar de aumentar o custo computacional dos treinos, a rede teria mais liberdade para fazer suas próprias inferências, o que talvez possa levar a um resultado melhor, especialmente associada às técnicas de *Deep Learning* (LECUN; BENGIO; HINTON, 2015) (ZHANG; WEIPINGLI; MO, 2018).

6.2.5 Recomendação de seleção de personagens

O fato de ser possível estimar qual o time vencedor em uma partida, ignorando completamente as habilidades dos jogares e utilizando apenas os personagens escolhidos, demonstra novamente a importância do processo de seleção. Sabendo quais combinações de personagens funcionam melhor contra outras, seria possível implementar um sistema que auxiliasse os jogadores no momento da seleção de personagens.

Considerando que acontecem anualmente centenas de campeonatos de DOTA 2 e que o prêmio para o maior destes campeonatos ultrapassou 25 milhões de dólares em 2018, uma ferramenta que consiga oferecer mais chances de vitória a um time seria de grande interesse e impacto.

Referências

ABADI, M. et al. Tensorflow: a system for large-scale machine learning. In: *OSDI*. [S.l.: s.n.], 2016. v. 16, p. 265–283. Citado na página 29.

ASCHER, D. et al. *Numerical python*. [S.l.]: Citeseer, 2001. Acesso em: 04 mai. 2018. Citado na página 30.

BONIFACIO, P. *How to Draft a Balanced Lineup in Dota 2 | FirstBlood®*. 2017. Disponível em: <https://firstblood.io/pages/blog/dota-2/how-to-draft-a-balanced-lineup-in-dota-2/>. Acesso em: 03 out. 2018. Citado na página 74.

BONIFACIO, P. *Who is the Best Dota 2 Player in the World? | FirstBlood®*. 2018. Disponível em: <https://firstblood.io/pages/blog/dota-2/who-is-the-best-dota-2-player-in-the-world/>. Acesso em: 05 out. 2018. Citado na página 74.

BRAGA, A. d. P.; CARVALHO, A.; LUDERMIR, T. B. *Redes neurais artificiais: teoria e aplicações*. [S.l.]: Livros Técnicos e Científicos Rio de Janeiro, 2000. Citado nas páginas 20, 21, 22, 25 e 66.

BROUWER, R. K. A hybrid neural network for input that is both categorical and quantitative. *International journal of intelligent systems*, Wiley Online Library, v. 19, n. 10, p. 979–1001, 2004. Citado na página 37.

BROWNLEE, J. *One-Hot Encode Data in Machine Learning?* 2017. Disponível em: <https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/>. Acesso em: 05 out. 2018. Citado na página 78.

CERRATO, N. *What is “the meta”? – Gamoloco Blog*. 2016. Disponível em: <https://blog.gamoloco.com/what-is-the-meta-4c79d47d6e98>. Acesso em: 01 set. 2018. Citado na página 72.

CHOLLET, F. et al. *Keras*. 2015. Disponível em: <https://keras.io>. Acesso em: 02 set. 2018. Citado na página 29.

CONLEY, K.; PERRY, D. How does he saw me? a recommendation engine for picking heroes in dota 2. *Np, nd Web*, v. 7, 2013. Citado nas páginas 30, 71, 74 e 77.

CUI, A.; CHUNG, H.; HANSON-HOLTRY, N. Opendota - all matches from march 2016 - matches. 2016. Disponível em: <opendota.com>. Citado na página 35.

CYBENKO, G. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, Springer, v. 2, n. 4, p. 303–314, 1989. Citado na página 22.

DEAN, P. *The Story of DOTA - How a bastard mod became its own genre*. 2011. Disponível em: <https://www.eurogamer.net/articles/2011-08-16-the-story-of-dota-article>. Acesso em: 25 out. 2015. Citado na página 18.

DHEERU, D.; TANISKIDOU, E. K. *UCI Machine Learning Repository: Iris Data Set*. 1988. Disponível em: <https://archive.ics.uci.edu/ml/datasets/Iris>. Acesso em: 14 jun. 2018. Citado na página 39.

DIAZ, G. I. et al. An effective algorithm for hyperparameter optimization of neural networks. *IBM Journal of Research and Development*, IBM, v. 61, n. 4, p. 9–1, 2017. Citado na página 42.

DOTAEDGE. *DotA Edge - Defence of the all pick*. 2015. Disponível em: <https://www.dotaedge.com>. Acesso em: 26 out. 2015. Citado na página 31.

ELLIS, D. *DotA Team*. 2013. Disponível em: <http://dotateam.me>. Acesso em: 27 out. 2015. Citado na página 32.

FISHER, R. A. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, Wiley Online Library, v. 7, n. 2, p. 179–188, 1936. Citado na página 39.

FREUND, Y.; SCHAPIRE, R. E. Large margin classification using the perceptron algorithm. *Machine Learning*, v. 37, n. 3, p. 277–296, Dec 1999. ISSN 1573-0565. Disponível em: <https://doi.org/10.1023/A:1007662407062>. Citado na página 21.

GEORGE, M. *Xavier Initialization · Manas George*. 2017. Disponível em: <https://mnsgrg.com/2017/12/21/xavier-initialization/>. Acesso em: 14 jun. 2018. Citado na página 77.

GÉRON, A. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. [S.l.]: "O'Reilly Media, Inc.", 2017. Citado nas páginas 19, 20, 21, 23, 24, 25, 26, 27, 28, 42, 43 e 66.

GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. [S.l.: s.n.], 2010. p. 249–256. Citado nas páginas 25 e 77.

GLOROT, X.; BORDES, A.; BENGIO, Y. Deep sparse rectifier neural networks. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. [S.l.: s.n.], 2011. p. 315–323. Citado nas páginas 25, 26, 27 e 28.

GODEC, K. *DOTA 2 Guide*. 2015. Disponível em: <http://purgegamers.true.io/g/dota-2-guide>. Acesso em: 24 nov. 2015. Citado nas páginas 16, 17, 18, 19 e 74.

HOWARD, N. *Paradoxes of rationality: theory of metagames and political behavior*. [S.l.]: MIT press, 1971. v. 1. Citado na página 72.

JACKSON, L. *Alliance Thrills In The International 2013 Win*. 2013. Disponível em: <https://www.redbull.com/us-en/alliance-thrills-in-the-international-2013-win>. Acesso em: 14 jun. 2018. Citado na página 74.

JOSHI, P. *Understanding Xavier Initialization In Deep Neural Networks | PERPETUAL ENIGMA*. 2016. Disponível em: <https://prateekvjoshi.com/2016/03/29/understanding-xavier-initialization-in-deep-neural-networks/>. Acesso em: 14 jun. 2018. Citado na página 77.

KERAS. *FAQ - Keras Documentation*. 2018. Disponível em: <https://keras.io/getting-started/faq/#what-does-sample-batch-epoch-mean>. Acesso em: 03 dez. 2018. Citado na página 66.

KESKAR, N. S. et al. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016. Citado na página 65.

KINKADE, N. Dota 2 win prediction. In: . [S.l.: s.n.], 2015. Citado nas páginas 16 e 19.

LAM, S. K.; PITROU, A.; SEIBERT, S. Numba: A llvm-based python jit compiler. In: ACM. *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. [S.l.], 2015. p. 7. Citado na página 30.

LAUB BRIAN RETFORD, R. E. F. et al. *plaidml: PlaidML is a framework for making deep learning work everywhere.* 2017. Disponível em: <https://github.com/plaidml/plaidml>. Acesso em: 04 mai. 2018. Citado na página 29.

LEAHY, B. *Surviving the basics of DOTA 2*. 2012. Disponível em: <https://www.engadget.com/2012/11/08/surviving-the-basics-of-dota-2/>. Acesso em: 25 out. 2015. Citado na página 18.

LEARNING, G. M. *Embeddings: Categorical Input Data | Machine Learning Crash Course | Google Developers*. 2018. Disponível em: <https://developers.google.com/machine-learning/crash-course/embeddings/categorical-input-data>. Acesso em: 07 out. 2018. Citado na página 78.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *nature*, Nature Publishing Group, v. 521, n. 7553, p. 436, 2015. Citado na página 78.

LIQUIPEDIA. *ppd - Liquipedia Dota 2 Wiki*. 2014. Disponível em: <https://liquipedia.net/dota2/Ppd>. Acesso em: 05 out. 2018. Citado na página 74.

LO, H. Z.; COHEN, J. P. Academic torrents: Scalable data distribution. *CoRR*, abs/1603.04395, 2016. Disponível em: <http://arxiv.org/abs/1603.04395>. Citado na página 35.

MATHWORKS. *What is MATLAB? - MATLAB & Simulink*. 2017. Disponível em: <https://www.mathworks.com/discovery/what-is-matlab.html>. Acesso em: 25 set. 2017. Citado na página 29.

MATHWORKS, I. T. *Improve Shallow Neural Network Generalization and Avoid Overfitting - MATLAB & Simulink*. 2018. Disponível em: <https://www.mathworks.com/help/deeplearning/ug/improve-neural-network-generalization-and-avoid-overfitting.html;jsessionid=0068ec560d90226c7817fb8e344f>. Acesso em: 03 dez. 2018. Citado na página 28.

MIND, T. *ReLU - Machine Learning for Humans - TinyMind*. 2018. Disponível em: <https://www.tinymind.com/learn/terms/relu>. Acesso em: 06 dez. 2018. Citado na página 27.

MITCHELL, T. M. et al. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, v. 45, n. 37, p. 870–877, 1997. Citado na página 19.

MURESAN, R. *5 Formidable Dota 2 Captains and Their Leadership Style*. 2018. Disponível em: <https://www.esports.net/top-formidable-dota-2-captains/>. Acesso em: 05 out. 2018. Citado na página 74.

NESTEROV, Y. E. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In: *Dokl. Akad. Nauk SSSR*. [S.l.: s.n.], 1983. v. 269, p. 543–547. Citado na página 25.

NG, C. *Vertex.AI - Announcing PlaidML: Open Source Deep Learning for Every Platform*. 2017. Disponível em: <http://vertex.ai/blog/announcing-plaidml>. Acesso em: 05 mai. 2018. Citado na página 29.

O'CONNOR, B. *tanh is a rescaled logistic sigmoid function | AI and Social Science – Brendan O'Connor*. 2013. Disponível em: <https://brenocon.com/blog/2013/10/tanh-is-a-rescaled-logistic-sigmoid-function/>. Acesso em: 05 mai. 2018. Citado na página 26.

OGUNMOLU, L. *Lekan's answer to What are the benefits of a tanh activation function over a standard sigmoid activation function for artificial neural nets, and vice versa? - Quora*. 2017. Disponível em: <https://www.quora.com/What-are-the-benefits-of-a-tanh-activation-function-over-a-standard-sigmoid-activation-function-for-ar answer/Lekan-4>. Acesso em: 07 out. 2018. Citado na página 26.

OLIPHANT, T. E. *A guide to NumPy*. [S.l.]: Trelgol Publishing USA, 2006. v. 1. Citado na página 30.

OPENDOTA. *December 2015 Data Dump*. 2015. Disponível em: <https://blog.opendota.com/2015/12/20/datadump/>. Acesso em: 05 mai. 2018. Citado na página 35.

PYTHON. *What is Python? Executive Summary*. 2018. Disponível em: <https://www.python.org/doc/essays/blurb/>. Acesso em: 25 abr. 2017. Citado na página 30.

ROSENBLATT, F. Perceptron simulation experiments. *Proceedings of the IRE*, IEEE, v. 48, n. 3, p. 301–309, 1960. Citado na página 20.

SAMUEL, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, IBM, v. 3, n. 3, p. 210–229, 1959. Citado na página 19.

SCIENCE, T. D. *Derivative of the Sigmoid function – Towards Data Science*. 2018. Disponível em: <https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e>. Acesso em: 06 dez. 2018. Citado na página 25.

SHARMA, S. *Activation Functions: Neural Networks – Towards Data Science*. 2017. Disponível em: <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>. Acesso em: 06 out. 2018. Citado na página 26.

SMITH, S. L. et al. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017. Citado na página 66.

SMITH, S. W. et al. The scientist and engineer's guide to digital signal processing. California Technical Pub. San Diego, 1997. Citado na página 46.

SNOEK, J.; LAROCHELLE, H.; ADAMS, R. P. Practical bayesian optimization of machine learning algorithms. In: PEREIRA, F. et al. (Ed.). *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012. p. 2951–2959. Disponível em: <http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf>. Citado na página 42.

SRIVASTAVA, N. et al. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, JMLR. org, v. 15, n. 1, p. 1929–1958, 2014. Citado nas páginas 28 e 29.

STRATZ. *Stratz eSports*. 2018. Disponível em: <https://stratz.com/en-us/>. Acesso em: 14 nov. 2018. Citado na página 73.

SUPERDATA. *Can Blizzard's Storm tackle a saturating MOBA market?* 2015. Disponível em: <https://www.superdataresearch.com/can-blizzards-storm-tackle-a-saturating-moba-market/>. Acesso em: 01 set. 2015. Citado na página 16.

SUPERDATA. *SuperData Research | Games data and market research » Market Brief — 2017 Digital Games & Interactive Media Year in Review*. 2017. Disponível em: <https://www.superdataresearch.com/market-data/market-brief-year-in-review/>. Acesso em: 04 set. 2018. Citado na página 16.

TECHLABS, M. *Is Python the most popular language for data science? - Maruti Techlabs*. 2016. Disponível em: <https://www.marutitech.com/python-data-science/>. Acesso em: 06 dez. 2018. Citado na página 30.

VALVE, C. *Dota 2 - International*. 2018. Disponível em: <http://www.dota2.com/international/overview>. Acesso em: 04 out. 2018. Citado na página 16.

VALVE, C. *Game Versions - Dota 2 Wiki*. 2018. Disponível em: <https://dota2.gamepedia.com/Game_Versions>. Acesso em: 05 out. 2018. Citado na página 36.

VALVE, C. *Heroes*. 2018. Disponível em: <http://www.dota2.com/heroes/>. Acesso em: 12 set. 2018. Citado nas páginas 16 e 18.

VIKTOROVICH, N. *True Picker*. 2015. Disponível em: <http://truepicker.com>. Acesso em: 20 out. 2015. Citado na página 31.

WANG, S. The unpredictability of standard back propagation neural networks in classification applications. *Management Science*, INFORMS, v. 41, n. 3, p. 555–559, 1995. ISSN 00251909, 15265501. Disponível em: <http://www.jstor.org/stable/2632981>. Citado na página 77.

WERBOS, P. J. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, IEEE, v. 78, n. 10, p. 1550–1560, 1990. Citado na página 20.

WIKIPEDIA. *Ramp function - Wikipedia*. 2018. Disponível em: <https://en.wikipedia.org/wiki/Ramp_function>. Acesso em: 06 dez. 2018. Citado na página 27.

WIKIPEDIA. *Tangente hiperbólica - Wikipedia, la enciclopedia libre*. 2018. Disponível em: <https://es.wikipedia.org/wiki/Tangente_hiperb%C3%B3lica>. Acesso em: 06 dez. 2018. Citado na página 26.

WILSON, D. R.; MARTINEZ, T. R. The general inefficiency of batch training for gradient descent learning. *Neural Networks*, Elsevier, v. 16, n. 10, p. 1429–1451, 2003. Citado na página 65.

XU, B. et al. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015. Citado na página 27.

XU, S.; CHEN, L. A novel approach for determining the optimal number of hidden layer neurons for fnn's and its application in data mining. 2008. Citado na página 42.

YIN-POOLE, W. *Dota 2 Preview - It's Valve Time*. 2011. Disponível em: <http://eurogamer.net/artivles/2011-08-19-dota-2-preview.com>. Acesso em: 25 out. 2015. Citado na página 18.

ZHANG, R.; WEIPINGLI; MO, T. Review of deep learning. *arXiv preprint arXiv:1804.01653*, 2018. Citado na página 78.

Apêndices

APÊNDICE A – Códigos SQL utilizados para inferência de dados estatísticos

Código para cálculo da sinergia crua de um herói com outro.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `hero_v_hero_syn_calc_atom`(OUT winprc FLOAT, heroA SMALLINT, heroB SMALLINT)
BEGIN

SELECT wins/total AS 'winrate' FROM (
    SELECT COUNT(*) AS 'total', COUNT(CASE WHEN
      (radiant_win='t'
      AND
      (hero1=heroB OR hero2=heroB OR hero3=heroB OR hero4=heroB OR hero0=heroB))
    OR
       (radiant_win='f' AND (hero5=heroA OR hero6=heroA OR hero7=heroA OR hero8=heroA OR hero9=heroA))
    THEN 1 END) AS 'wins'
    FROM (
        SELECT m.radiant_win, m.hero0, m.hero1, m.hero2, m.hero3, m.hero4, m.hero5, m.hero6, m.hero7, m.hero8, m.hero9 FROM
        matches_high_skill m WHERE
        (
          (m.hero0=heroA OR m.hero1=heroA OR m.hero2=heroA OR m.hero3=heroA OR m.hero4=heroA)
          AND
          (m.hero1=heroB OR m.hero2=heroB OR m.hero3=heroB OR m.hero4=heroB OR m.hero0=heroB)
        )
        OR
        (
          (m.hero5=heroA OR m.hero6=heroA OR m.hero7=heroA OR m.hero8=heroA OR m.hero9=heroA)
          AND
          (m.hero5=heroB OR m.hero6=heroB OR m.hero7=heroB OR m.hero8=heroB OR m.hero9=heroB)
        )
    ) AS subsubquery
)
AS subquery INTO winprc;
END
```

Código para cálculo da taxa média geral de vitória de um personagem.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `populate_winrate`()
BEGIN
DECLARE i INT DEFAULT 0;
WHILE i < 112 DO
  SET i = i + 1;
  SET @times = (
    SELECT
      count( * )
    FROM
      matches_high_skill m
    WHERE(
       (
        m.hero0 = i OR m.hero1 = i OR m.hero2 = i OR m.hero3=i OR m.hero4=i
      )
      AND m.radiant_win = 't'
    )
    OR (
      (
        m.hero5 = i OR m.hero6 = i OR m.hero7 = i OR m.hero8=i OR m.hero9=i
      )
      AND m.radiant_win = 'f'
    )
  ) / ( SELECT count( * )
    FROM
      matches_high_skill m
    WHERE(
      m.hero0 = i OR m.hero1 = i OR m.hero2 = i OR m.hero3=i OR m.hero4=i
    )
    OR (
      m.hero5 = i OR m.hero6 = i OR m.hero7 = i OR m.hero8=i OR m.hero9=i
    )
    );
  UPDATE hero_winrate SET avg_winrate = @times WHERE hero_id = i;
END WHILE;
END
```

Código para cálculo da taxa média de vitória de um herói para determinada faixa de tempo.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `calc_winrate_by_time_atom` (OUT
    winprc FLOAT, tempoA SMALLINT, tempoB SMALLINT, i tinyint)
BEGIN

SELECT wins/total AS 'winrate' FROM (
   SELECT
      count( * ) as 'total', COUNT(CASE WHEN
      (
         ( ( m.hero0 = i OR m.hero1 = i OR m.hero2 = i OR m.hero3 = i OR m.
            hero4 = i) AND m.radiant_win = 't' )
         OR
         ( ( m.hero5 = i OR m.hero6 = i OR m.hero7 = i OR m.hero8 = i OR m.
            hero9 = i) AND m.radiant_win = 'f' )
      ) THEN 1 END ) as 'wins'
   FROM
      matches_high_skill m
   WHERE
   (
      ((
         (
            m.hero0 = i
            OR m.hero1 = i
            OR m.hero2 = i
            OR m.hero3 = i
            OR m.hero4 = i
         )
      )
      OR
      (
         (
            m.hero5 = i
            OR m.hero6 = i
            OR m.hero7 = i
            OR m.hero8 = i
            OR m.hero9 = i
         )
      ))
      AND duration <= tempoB AND duration >= tempoA
   )
) as subquery
INTO winprc;
END
```

Código para cálculo da vantagem crua de um herói sobre outro.

```
1  CREATE DEFINER=`root`@`localhost` PROCEDURE `insert_advantage_hero_v_hero`()
2  BEGIN
3  DECLARE i INT DEFAULT 0;
4  DECLARE j INT DEFAULT 0;
5
6  WHILE  i < 112 DO
7
8      SET i = i + 1;
9
10     IF ((i = 24) OR (i = 108)) THEN
11       SET i = i + 1;
12     END IF;
13
14     SET j = i;
15
16     WHILE  j < 112 DO
17         SET j = j + 1;
18
19         IF ((j = 24) OR (j = 108)) THEN
20           SET j = j + 1;
21         END IF;
22
23         CALL total_games_count ( @qtd_total, i, j );
24         CALL total_games_win ( @qtd_win, i, j );
25         INSERT INTO hero_v_hero VALUES ( i, j, NULL, (@qtd_win / @qtd_total
               ) ) ON DUPLICATE KEY UPDATE hero_v_hero.vantagem = (@qtd_win /
               @qtd_total) ;
26     END WHILE;
27 END WHILE;
28 END
```

Código para normalização da vantagem crua de um herói sobre outro, fornecendo a vantagem relativa de um herói sobre o outro.

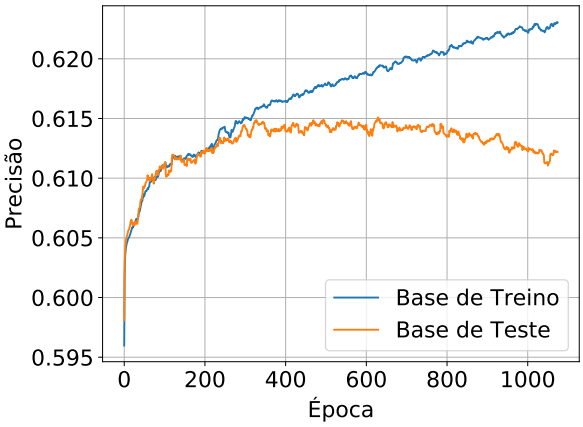
```
1
2  CREATE DEFINER=`root`@`localhost` PROCEDURE `
       calc_insert_normalizacao_vantagem`()
3  BEGIN
4
5  DECLARE i INT DEFAULT 0;
6  DECLARE j INT DEFAULT 0;
7
8  WHILE  i < 112 DO
9
10     SET i = i + 1;
11     SET j = i;
12
13     WHILE  j < 112 DO
14         SET j = j + 1;
15
16         select avg_winrate from hero_winrate where hero_id = i into @avg;
17
18         select vantagem from hero_v_hero where hero_id = i and
               target_hero_id = j into @vnt;
19
20         update hero_v_hero_normalized SET vantagem = (@vnt-@avg) where
               hero_v_hero_normalized.hero_id = i AND hero_v_hero_normalized.
               target_hero_id = j;
21
22     END WHILE;
23 END WHILE;
24
25 END
```

Código para normalização da sinergia crua de um herói com outro, fornecendo a sinergia relativa.

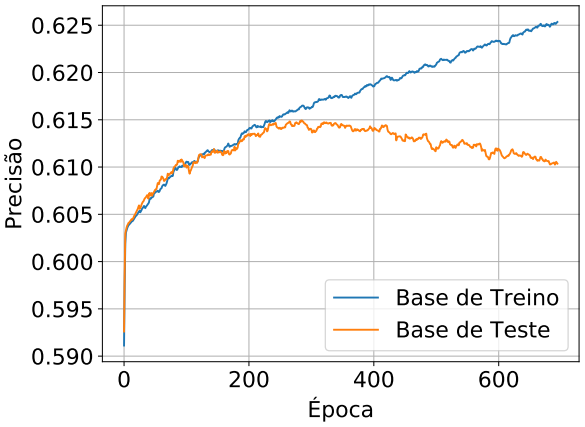
```
1  CREATE DEFINER=`root`@`localhost` PROCEDURE `normalize_sinergia`()
2  BEGIN
3  DECLARE i INT DEFAULT 0;
4  DECLARE j INT DEFAULT 0;
5  WHILE  i < 112 DO
6
7      SET i = i + 1;
8      SET j = i;
9
10     WHILE  j < 112 DO
11         SET j = j + 1;
12
13         SELECT avg_winrate from hero_winrate h WHERE h.hero_id = i INTO
               @avg_wr_hero1;
14         SELECT avg_winrate from hero_winrate h WHERE h.hero_id = j INTO
               @avg_wr_hero2;
15         SELECT sinergia from hero_v_hero hvh WHERE hvh.hero_id = i AND hvh.
               target_hero_id = j INTO @raw_sinnergy;
16
17         update hero_v_hero_normalized hvh SET sinergia = ( @raw_sinnergy -
               ((@avg_wr_hero1 + @avg_wr_hero2) / 2 ) ) WHERE hvh.hero_id = i
               AND hvh.target_hero_id = j;
18
19     END WHILE;
20 END WHILE;
21 END
```

APÊNDICE B – Comportamento individual das diferentes configurações de camada da rede em cada um cenários E, F, G, H, I e J.

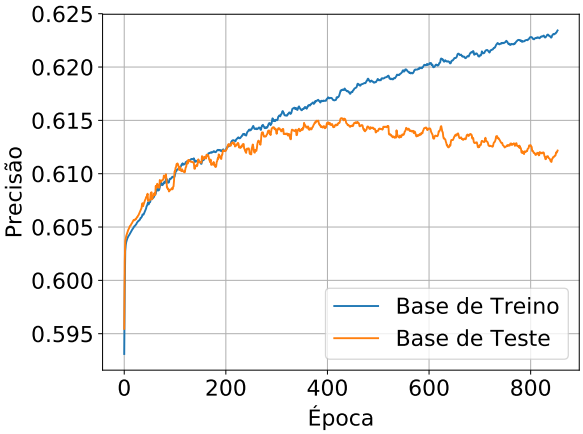
Figura 38 – Comportamento individual dos treinamentos realizados no Cenário E. Função de ativação tangente hiperbólica, **sem** utilização de *Dropout*



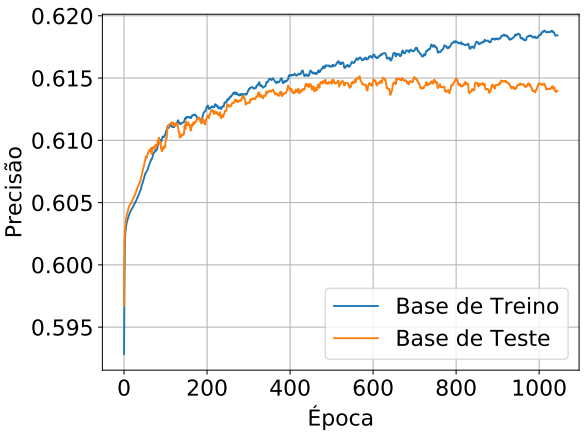
(a) 200-50



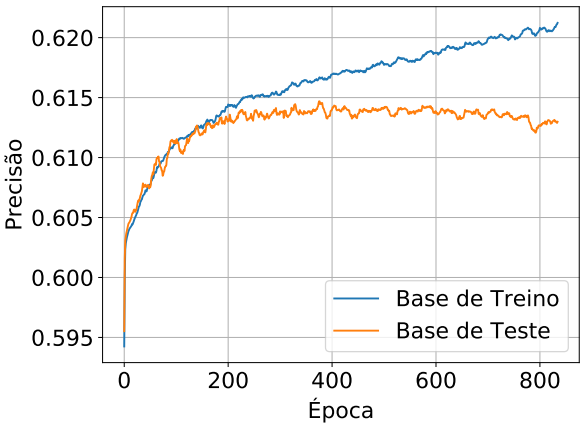
(b) 200-100-50



(c) 300-75



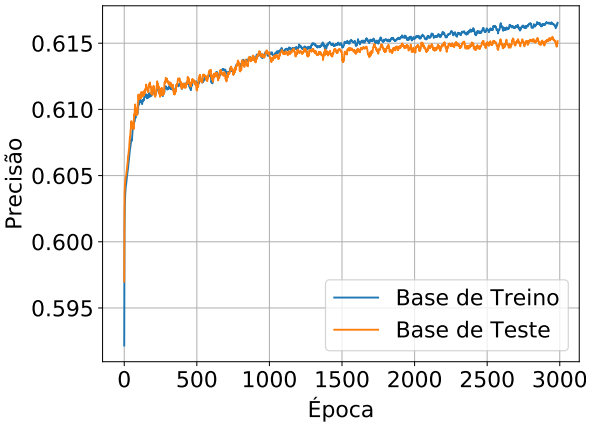
(d) 100-25



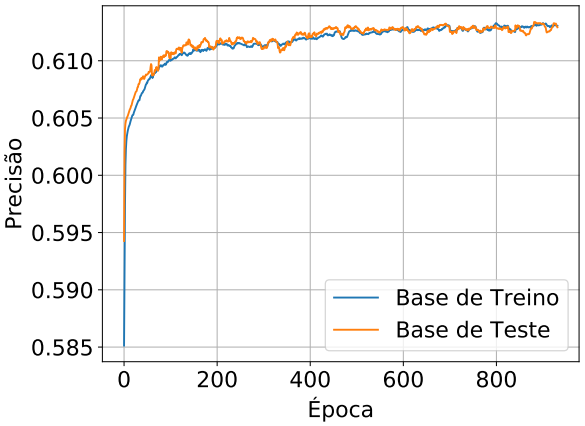
(e) 100-50-25-10

Fonte – Elaborada pelo autor.

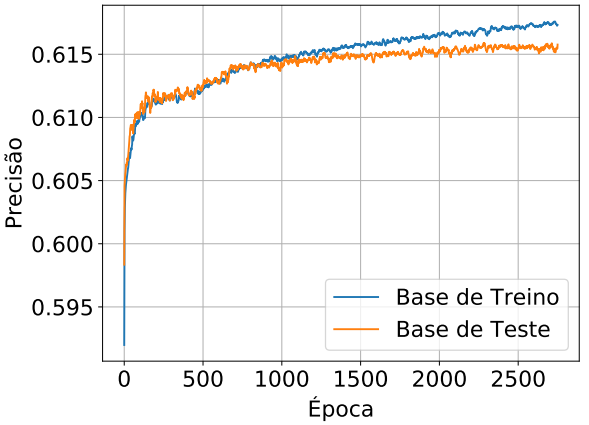
Figura 39 – Comportamento individual dos treinamentos realizados no Cenário F. Função de ativação tangente hiperbólica, **com** utilização de *Dropout*



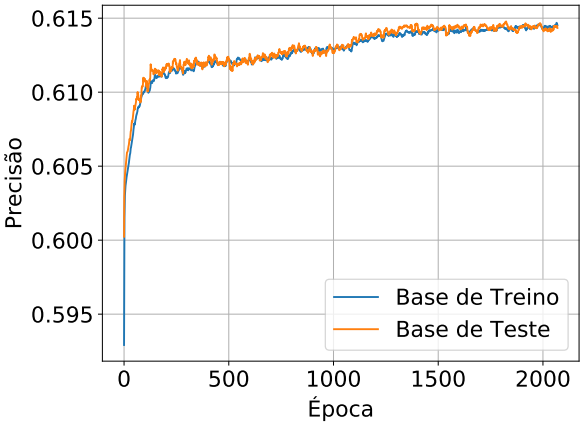
(a) 200-50



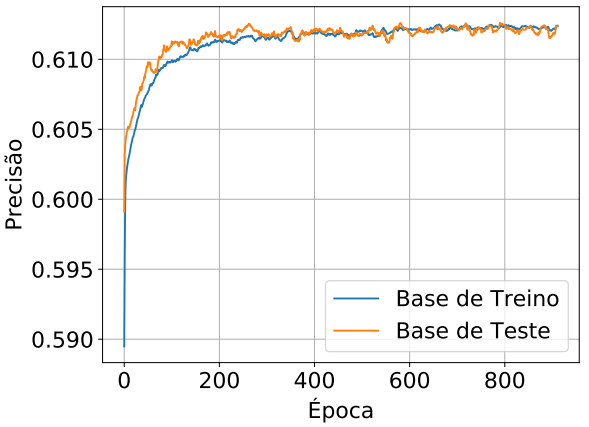
(b) 200-100-50



(c) 300-75



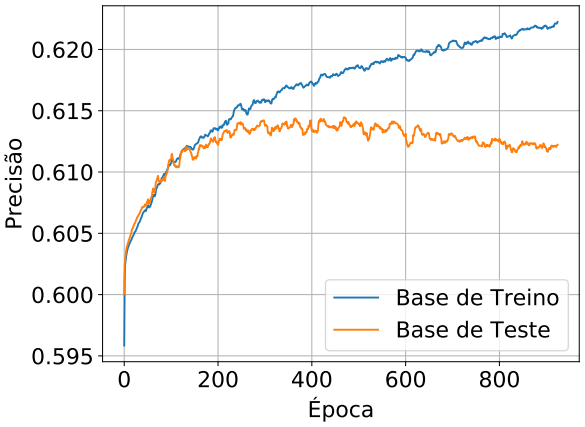
(d) 100-25



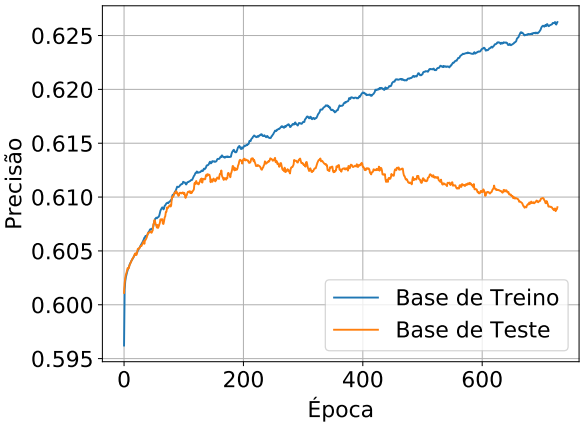
(e) 100-50-25-10

Fonte – Elaborada pelo autor.

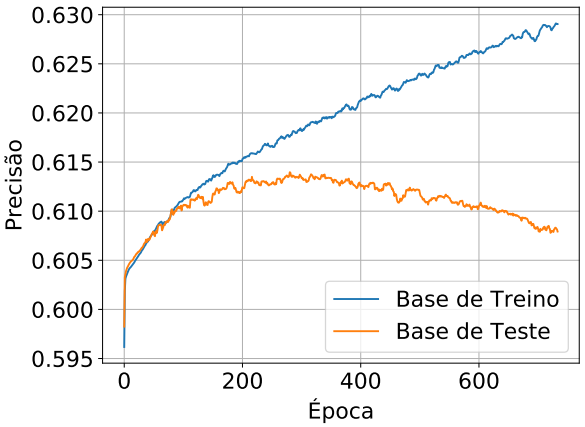
Figura 40 – Comportamento individual dos treinamentos realizados no Cenário G. Função de ativação ReLU, **sem** utilização de *Dropout*



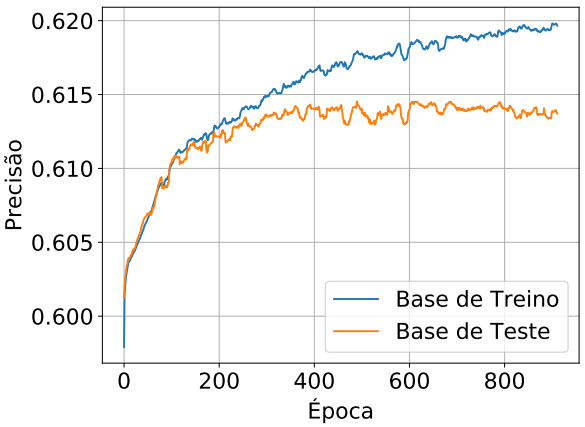
(a) 200-50



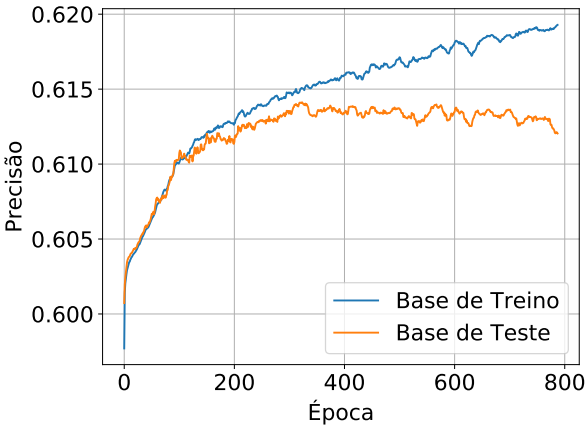
(b) 200-100-50



(c) 300-75



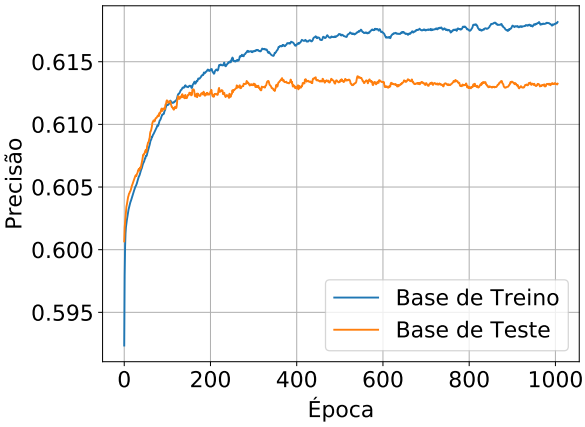
(d) 100-25



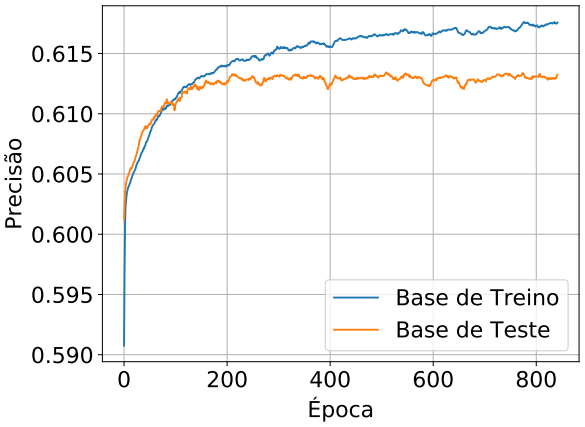
(e) 100-50-25-10

Fonte – Elaborada pelo autor.

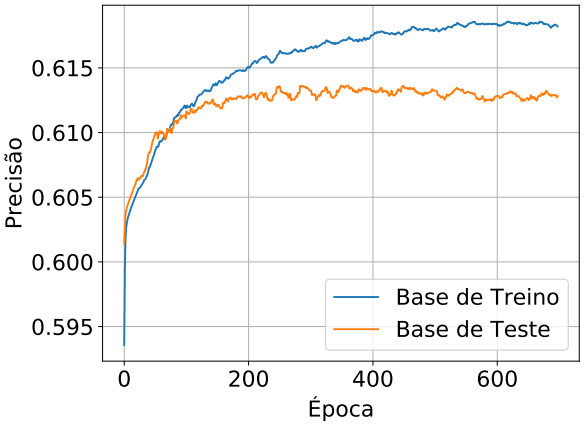
Figura 41 – Comportamento individual dos treinamentos realizados no Cenário H. Função de ativação ReLU, **com** utilização de *Dropout*



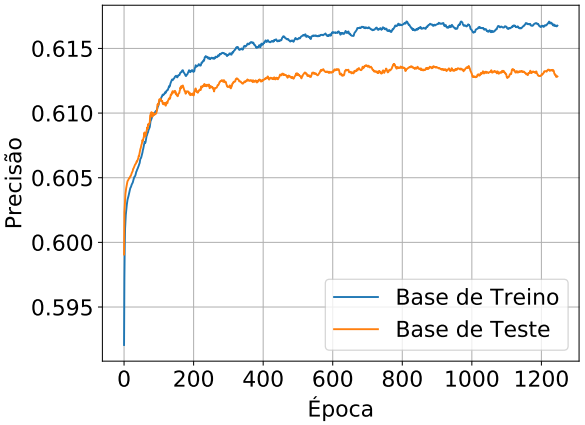
(a) 200-50



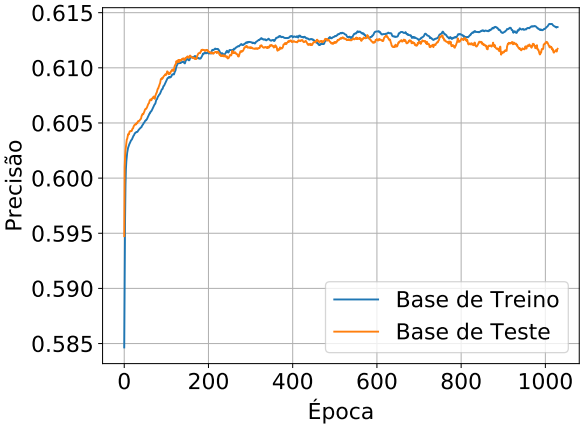
(b) 200-100-50



(c) 300-75



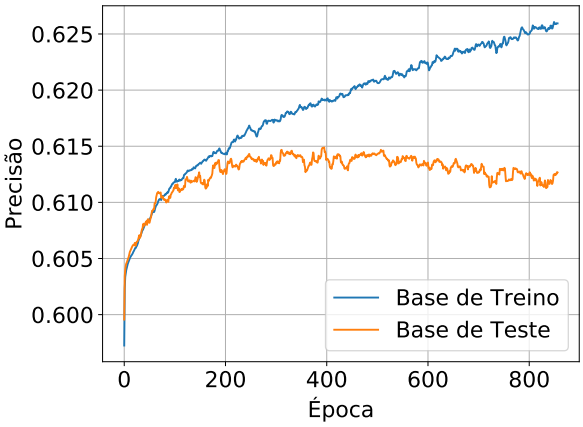
(d) 100-25



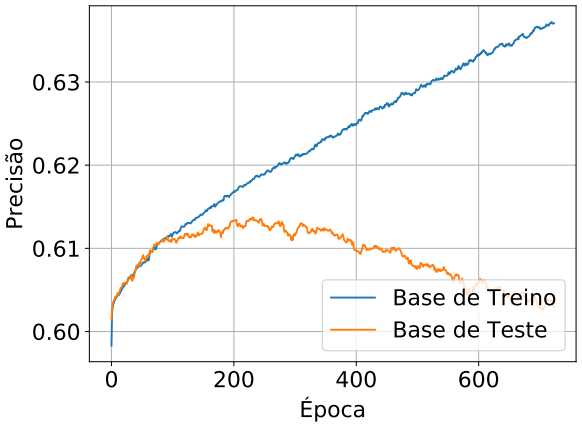
(e) 100-50-25-10

Fonte – Elaborada pelo autor.

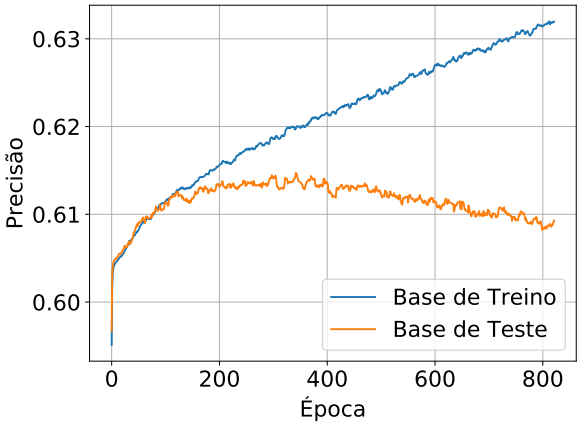
Figura 42 – Comportamento individual dos treinamentos realizados no Cenário I. Função de ativação Leaky ReLU, **sem** utilização de *Dropout*



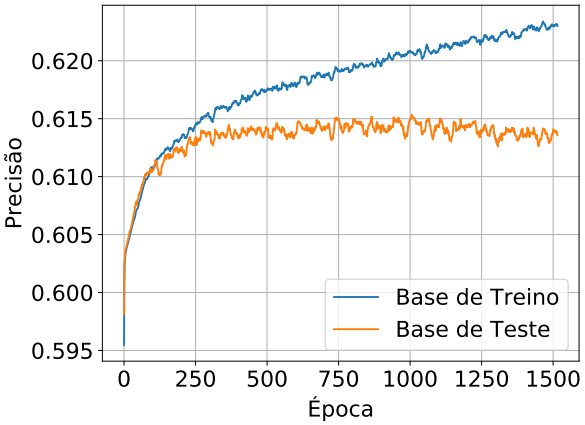
(a) 200-50



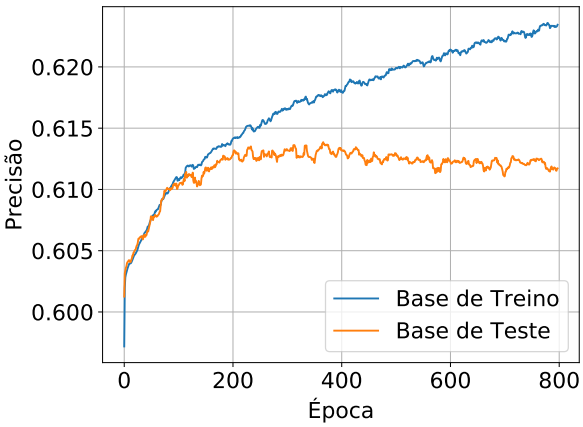
(b) 200-100-50



(c) 300-75



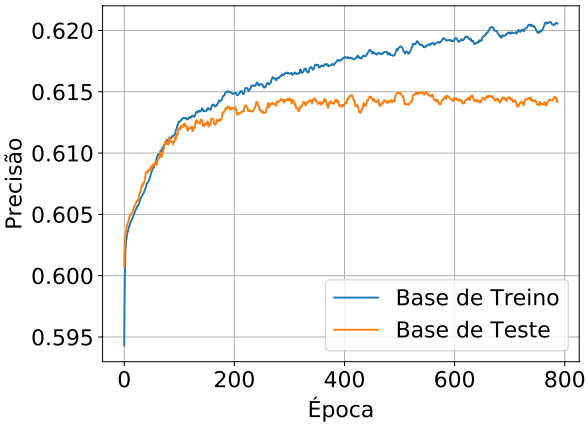
(d) 100-25



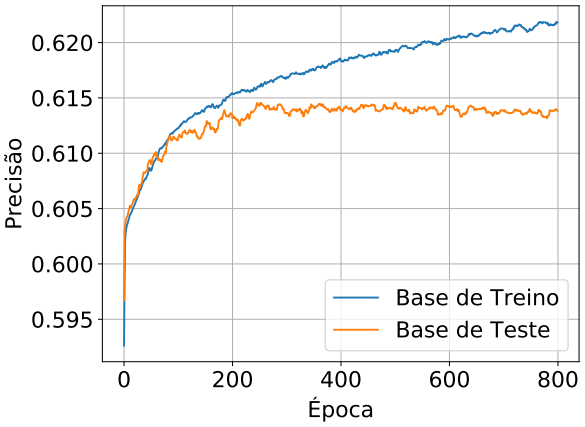
(e) 100-50-25-10

Fonte – Elaborada pelo autor.

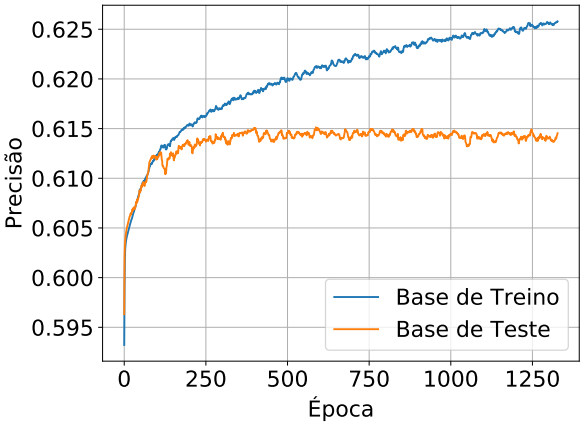
Figura 43 – Comportamento individual dos treinamentos realizados no Cenário J. Função de ativação Leaky ReLU, **com** utilização de *Dropout*



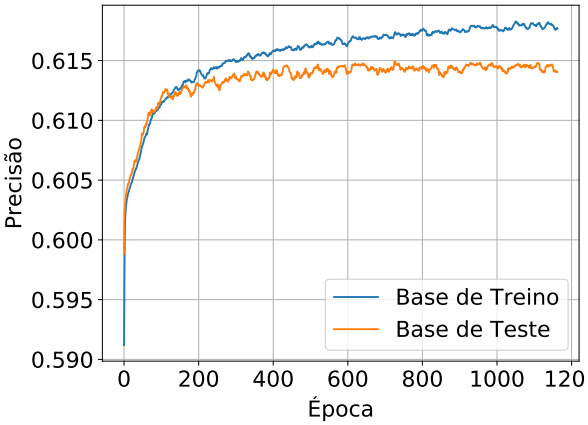
(a) 200-50



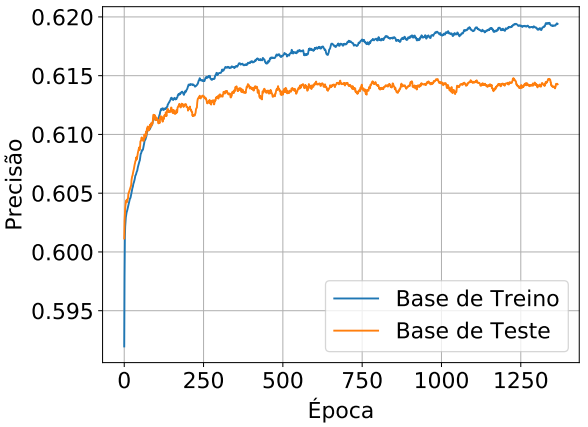
(b) 200-100-50



(c) 300-75



(d) 100-25



(e) 100-50-25-10

Fonte – Elaborada pelo autor.

Anexos

ANEXO A – Exemplo de formatação dos registros do OpenDota

```
[{// Data dumps are arrays of match objects
   "match_id": 2000594819,
   "match_seq_num": 1764515493,
   "radiant_win": false,
   "start_time": 1450027278,
   "duration": 1089,// In seconds
   "tower_status_radiant": 1975,// Bitmask, see here https://wiki.
      teamfortress.com/wiki/WebAPI/GetMatchDetails#Tower_Status
   "tower_status_dire": 2047,
   "barracks_status_radiant": 63,
   "barracks_status_dire": 63,
   "cluster": 133,
   "first_blood_time": 203,
   "lobby_type": 1,
   "human_players": 10,
   "leagueid": 4176,
   "positive_votes": 0,
   "negative_votes": 0,
   "game_mode": 2,
   "engine": 1,
   "parse_status": 2,
   "chat": [{
      "time": -89,
      "type": "chat", // Player message or could be event type, I.E "
         CHAT_MESSAGE_TOWER_KILL"
      "unit": "MAKES ME HAPPY", // Name of Player
      "key": "go?", // What they said
      "slot": 2// Their slot
   },... ],
   "version": 15,// Parse version, used internally for YASP
   "players": [{// Array of players
      "account_id": 223332951,// Standard WebAPI stuff
      "player_slot": 0,
      "hero_id": 26,
      "item_0": 46,
      "item_1": 36,
      "item_2": 38,
      "item_3": 0,
      "item_4": 0,
      "item_5": 188,
      "kills": 0,
      "deaths": 5,
```

```
      "assists": 1,
      "leaver_status": 0,
      "gold": 19,
      "last_hits": 3,
      "denies": 4,
      "gold_per_min": 110,
      "xp_per_min": 112,
      "gold_spent": 2085,
      "hero_damage": 1072,
      "tower_damage": 33,
      "hero_healing": 0,
      "level": 6,
      "ability_upgrades": [{
         "ability": 5044,
         "time": 774,
         "level": 1
      },... ],
      "additional_units": null,
      "stuns": 12.7967, // Total stun duration of all stuns by this user
      "max_hero_hit": {
         "type": "max_hero_hit",
         "time": 561,
         "max": true,
         "inflictor": "lion_impale",
         "unit": "npc_dota_hero_lion",
         "key": "npc_dota_hero_winter_wyvern",
         "value": 105,
         "slot": 0,
         "player_slot": 0
      },
      "times": [0,60,120,180,240,300,360,420,480,540,600,660,720,780,840,900
         ,960,1020,1080,1140,1200,1260,1320,1380],
      "gold_t": [0,100,200,300,400,500,600,700,846,946,1046,1194,1294,1394,1
         494,1594,1784,1884,1984,2000,2000,2000,2000,2000],
      "lh_t": [0,0,0,0,0,0,0,0,1,1,1,2,2,2,2,2,3,3,3,3,3,3,3,3],
      "xp_t": [0,62,135,232,304,448,510,716,849,922,1273,1324,1397,1503,1582
         ,1582,1841,1841,2045,2045,2045,2045,2045,2045],
      "obs_log": [{
         "time": 880,
         "key": [114,92]// (x, y)
      },{
         "time": 980,
         "key": [102,116]
      }],
   },... ]
```