UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Marcelo Cargnelutti Rossato

REDES NEURAIS RECORRENTES LSTM E SUAS APLICAÇÕES

Santa Maria, RS
2018

Marcelo Cargnelutti Rossato

REDES NEURAIS RECORRENTES LSTM E SUAS APLICAÇÕES

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Engenharia de Controle e Automação**.

ORIENTADOR: Prof. Rodrigo da Silva Guerra

Santa Maria, RS
2018

Marcelo Cargnelutti Rossato

REDES NEURAIS RECORRENTES LSTM E SUAS APLICAÇÕES

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Engenharia de Controle e Automação**.

Aprovado em 10 de agosto de 2018:

Rodrigo da Silva Guerra, Dr. (UFSM)
(Presidente/Orientador)

Daniel Fernando Tello Gamarra, Dr. (UFSM)

Paulo Lilles Jorge Drews Junior, Dr. (FURG)

Santa Maria, RS
2018

AGRADECIMENTOS

À Universidade Federal de Santa Maria pelas oportunidades proporcionadas. Aos professores do Curso de Engenharia de Controle e Automação pela contribuição para o meu desenvolvimento pessoal e profissional. Ao professor Rodrigo Guerra pelo suporte e orientação na elaboração desse trabalho. A minha família e amigos pelo amor, incentivo e apoio incondicional.

The best way to predict your future is to create it

(Peter Drucker)

RESUMO

REDES NEURAIS RECORRENTES LSTM E SUAS APLICAÇÕES

AUTOR: Marcelo Cargnelutti Rossato
ORIENTADOR: Rodrigo da Silva Guerra

O objetivo deste trabalho é a revisão do modelo de rede neural artificial *Long Short-Term Memory* (LSTM) e a sua aplicação em diferentes atividades, com o intuito de avaliar sua versatilidade. São apresentados quatro estudos de caso, sendo um de reconhecimento e classificação de dígitos em tempo real e outros três de geração de dados, a saber: produção de fragmentos de texto, previsão de índices de poluição e realização de movimentos no jogo Pong. Foram desenvolvidos quatro modelos de rede neural recorrente LSTM, sendo implementadas em Python e treinadas por meio do algoritmo de *backpropagation through time*. A rede mostrou-se robusta em todas as atividades propostas e extremamente flexível, uma vez que eram necessárias poucas adaptações no modelo da rede neural para realizar uma aplicação diferente. Com base nos resultados obtidos, pode-se concluir que há um grande potencial de utilização da rede em diversas aplicações.

Palavras-chave: *Deep Learning*; LSTM; Redes Neurais Recorrentes.

ABSTRACT

LSTM RECURRENT NEURAL NETWORKS AND ITS APPLICATIONS

AUTHOR: Marcelo Cargnelutti Rossato
ADVISOR: Rodrigo da Silva Guerra

The objective of this work is to review the Long Short-Term Memory (LSTM) artificial neural network model and its application in different activities, in order to evaluate its versatility. Four activities are presented, one of which is the recognition and classification of digits in real time and other three of data generation, namely: production of text fragments, prediction of pollution indexes and realization of movements in the Pong game. Four models of the LSTM recurrent neural network were developed, being implemented in Python and trained through the algorithm of backpropagation through time. The network proved to be robust in all proposed scenarios and extremely flexible, since few adaptations were necessary in the neural network model to be applied to a different application. Based on the obtained results, it can be concluded that there is a great potential of use of the network in several applications.

Keywords: Deep Learning; LSTM; Recurrent Neural Networks.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Este trabalho possui como tema a utilização de redes neurais artificiais LSTM e apresenta possíveis aplicações das mesmas no reconhecimento dinâmico de dígitos em tempo real e na produção de informações pela rede, por meio da escrita de caracteres de texto, previsões de índices de poluição e realização de movimentos em um jogo de Pong.

O primeiro modelo de rede neural artificial foi proposto por McCulloch e Pitts (1943), abordando uma possível maneira de interpretar matematicamente o sistema nervoso, composto por diversos neurônios com seus axônios e dendritos, e como ocorrem as conexões (sinapses) entre esses neurônios. Essas redes neurais artificiais obtiveram um grande destaque devido ao seu bom desempenho e ganharam muita força por volta dos anos 80, porém começaram a cair em desuso após isso devido às limitações para treinamento de redes profundas.

Hinton, Osindero e Teh (2006) apresentaram uma nova abordagem que ficou mais tarde conhecida como *Deep Learning*, decompondo um modelo complexo como uma combinação de modelos mais simples e aprendidos de maneira sequencial. Essa abordagem, associada também ao desenvolvimento de hardwares mais potentes e de uma maior disponibilidade de dados para treinamento, possibilitou o ressurgimento e uma grande popularização das redes neurais nas mais diversas áreas, como classificações de objetos e sequências, identificação de imagens e reconhecimento de fala.

Atualmente, a utilização de algoritmos de *Deep Learning* foi responsável por grandes avanços tecnológicos e a rede neural recorrente *Long Short-Term Memory* (LSTM) é utilizada como base para diversos sistemas de grandes empresas, como sistemas de reconhecimento de voz da Apple, Google e Microsoft, além de estar presente também em assistentes pessoais e sistemas de tradução, como apresenta Schmidhuber (2017), um dos criadores da rede LSTM. Com isso, salienta-se a grande eficiência e aplicabilidade das redes neurais artificiais.

Diante de diversas possibilidades de aplicação, as atividades de reconhecimento de letras e dígitos manuscritos são um dos focos desse trabalho. Simard et al. (2003) mostram que a utilização de redes neurais artificiais foi a que obteve um melhor desempenho em uma tarefa de reconhecimento de caligrafia utilizando a database MNIST, que é comumente utilizada para avaliação de sistemas de processamento de imagens.

De maneira similar, Sinha (2002) implementou uma rede neural artificial para o reconhecimento de dígitos e aplicou a mesma no reconhecimento de valores escritos em cheques, contribuindo ainda mais para mostrar a versatilidade da rede uma vez que a mesma já foi usada em aplicações diferentes. Ressalta-se aqui a elevada eficiência das redes neurais artificiais quando comparadas com outros meios de reconhecimento de escritas manuais.

Entretanto, na vasta maioria dos trabalhos conhecidos, os dígitos são identificados através de suas imagens prontas. Nosso trabalho se diferencia por analisar todo o processo dinâmico de escrita, o qual apresenta uma riqueza de informação temporal a respeito da construção incremental que pode melhorar o desempenho do sistema e até permitir tarefas de reconhecimento mais desafiadoras (como a de verificar a autenticidade do autor de uma caligrafia, por exemplo).

De acordo com Brownlee (2016), por exemplo, além de serem aplicadas como modelos preditivos, como é o caso do reconhecimento de dígitos em tempo real, as redes neurais artificiais também podem ser usadas como modelos geradores, aprendendo a partir das sequências de dados introduzidos e gerando novas sequências plausíveis.

O autor apresenta também uma aplicação dos modelos geradores na construção de fragmentos de texto, de maneira semelhante a que será abordada ao longo deste trabalho. Para ilustrar a variedade de possibilidades do uso dessas redes neurais na geração de informações, serão também implementados exemplos para suas aplicações na previsão de índices de poluição e em movimentações no jogo Pong.

Isso posto, esse trabalho busca conciliar a abrangência de diferentes domínios de aplicações da LSTM, ainda assim mantendo um razoável nível de profundidade através de estudos de caso práticos.

1.1 OBJETIVOS

O objetivo desse trabalho é realizar a implementação e aplicação de redes neurais artificiais LSTM em diferentes conjuntos de dados para salientar a versatilidade da utilização das mesmas.

Dessa forma, busca-se implementar em Python modelos de redes neurais artificiais LSTM que, tendo como base um conjunto de dados adequado e pré-processado, sejam capazes de realizar as seguintes tarefas de identificação e de geração de informações:

- Reconhecer e classificar dígitos em tempo real
- Produzir fragmentos de texto
- Fazer previsões de níveis de poluição
- Realizar movimentos no jogo Pong

2 REVISÃO BIBLIOGRÁFICA

Ao longo deste capítulo, será abordado o referencial teórico utilizado como base para este trabalho. Na primeira seção, serão apresentados os principais conceitos necessários para o entendimento de redes neurais e desse trabalho como um todo. Na segunda seção, serão analisados o crescimento da popularidade das redes LSTM, suas aplicações mais comuns e alguns trabalhos relacionados com este.

2.1 REDES NEURAIS ARTIFICIAIS

As redes neurais artificiais têm seu funcionamento baseado no conceito biológico de uma rede de neurônios. Segundo Kovacs (2002), “sistemas biológicos possuem a essencial propriedade de serem capazes de aprender uma função. Poderia-se portanto imaginar uma maneira de ensinar a rede artificial até que esta aprendesse a função desejada. ”

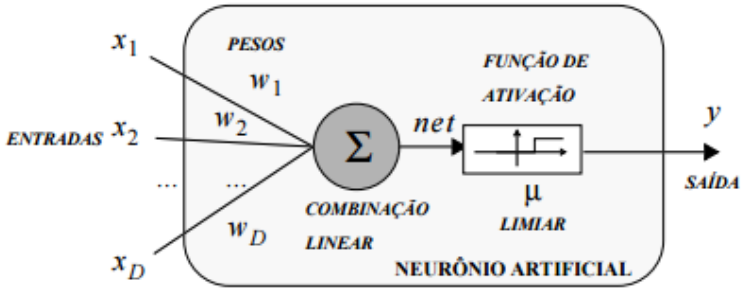
Assim, o treinamento de uma rede neural artificial pode ocorrer por meio de uma aprendizagem supervisionada, na qual o modelo recebe pares de informação, contendo a entrada e a saída desejada, como o desenho de dígitos e a sua identificação por exemplo, ou por aprendizagem não supervisionada, recebendo apenas as entradas e buscando padrões que ajudem a descrever os dados recebidos.

Além disso, as redes também podem ser classificadas em dois grandes tipos: classificação ou regressão. Em situações de classificação, as redes neurais artificiais buscam categorizar os dados de entrada em diversos grupos denominados classes. Já em situações de regressão, os dados não são mais separados em classes e a saída da rede neural passa a ser um valor numérico.

Uma rede neural artificial é composta por diversos neurônios, onde cada um deles recebe algumas informações de entrada e realiza uma combinação linear desses elementos, associando pesos à importância de cada uma dessas informações. Após essa combinação linear, é aplicada uma função de ativação, a qual geralmente é não linear.

De forma mais técnica, uma rede neural pode ter suas sinapses representadas como uma matriz, realizando o processamento das informações de entrada por meio de uma multiplicação de matrizes. Ainda, o resultado dessa combinação linear ou multiplicação é aplicado em uma função de ativação, resultando em um sinal de saída, como pode ser observado na Figura 2.1.

Figura 2.1 – Modelo de um neurônio

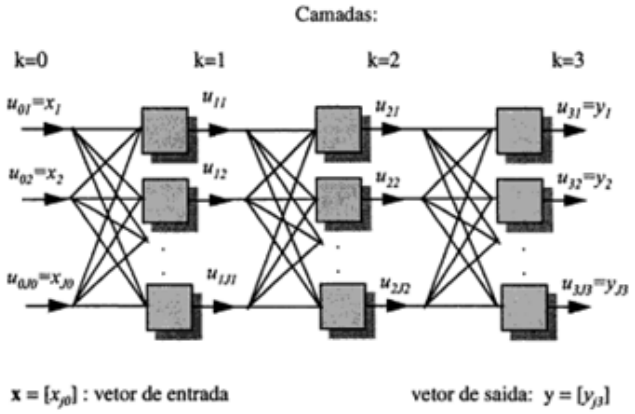


Fonte: (RAUBER, 2005)

Uma função de ativação comumente utilizada em situações de classificação é a função *softmax*, a qual normaliza a saída dos neurônios para valores compreendidos entre 0 e 1 e faz com que a soma de todos esses elementos seja igual a 1. Assim, ela possui como entrada um vetor da forma $x = (x_1, x_2, \ldots, x_n)$ e retorna como saída um vetor de mesmo comprimento $y = (y_1, y_2, \ldots, y_n)$, onde cada elemento do vetor de saída é definido por $y_i = \frac{e^{x_i}}{\sum e^{x_j}}$. A função *softmax* é de extrema importância, pois ela é a responsável por fazer com que o sinal de saída possa ser compreendido como um vetor de probabilidades.

Uma rede neural artificial é composta por vários neurônios e, possivelmente, por várias camadas que se conectam. Um modelo de rede neural artificial é apresentado na Figura 2.2. Assim, a primeira camada (k=0) representa os valores de entrada, as camadas k=1 e k=2 representam duas camadas escondidas e a última camada (k=3) representa os valores de saída.

Figura 2.2 – Modelo de uma rede neural artificial



Fonte: (KOVACS, 2002)

Como pode ser observado, as informações de entrada da rede são todas levadas aos neurônios da camada k=1, os quais realizam suas combinações lineares e repassam os sinais de saída para os neurônios da camada k=2, os quais repetem o processo e

repassam os sinais para os neurônios da camada k=3, que finalmente realizam novamente as combinações lineares e geram um sinal de saída do sistema. Ainda, essa saída costuma ser aplicada a uma função de ativação, como é o caso da *softmax*.

Cabe salientar que esse formato final da saída está intimamente relacionado com o formato esperado pela regressão ou com a quantidade de classificações que podem ser realizadas. No caso de classificação, ao receber um conjunto de dados de entrada, o primeiro elemento desse vetor de saída se refere à probabilidade da rede classificar esses dados com a primeira identificação possível, o segundo elemento refere-se à probabilidade de classificá-los com a segunda identificação, e assim sucessivamente.

Ao inserir um conjunto de dados de entrada que representam a segunda classificação, por exemplo, a saída desejada é de que a probabilidade é de 100% de os dados representarem a segunda classificação e de 0% de ser qualquer uma das outras, ou seja, o vetor de saída deveria ser da forma [0,1,0,. . . ,0]. Essa transformação de uma classe ou rótulo de um valor numérico para um vetor com um elemento unitário e os outros nulos é denominado *one-hot encoding*.

Com essa notação, é mais natural definir o erro que será denominado de *categorical crossentropy*. Ora, como a saída da rede pode ser vista como um vetor de probabilidades e a saída desejada usualmente é um vetor no formato *one-hot encoding*, o erro é definido como o erro quadrático desses dois vetores. Assim, se a saída obtida for o vetor $v = (v_1, v_2, \ldots, v_n)$ e a saída desejada for o vetor $u = (u_1, u_2, \ldots, u_n)$, o erro é definido da seguinte forma: $Erro^2 = \sum (v_i - u_i)^2$.

No que se refere ao aprendizado e implementação de redes neurais artificiais, para realizar o processo de treinamento supervisionado da rede, além de definir a estrutura da mesma, é necessário que se possua um banco de dados. Esses dados devem ser separados em três grupos: treinamento, validação e teste.

Os dados de treinamento são os responsáveis pelo aprendizado da rede e nos quais será aplicado o método mencionado a seguir. Os dados de validação são utilizados ao longo do treinamento para verificar se a rede está obtendo um bom desempenho no reconhecimento de outros sinais de entrada e evitar que ela simplesmente memorize os dados de treinamento. Por fim, os dados de teste são utilizados após o processo de treinamento para verificar se o treinamento produziu um resultado satisfatório, observando a robustez da rede neural treinada por meio da precisão e do erro da mesma nesse conjunto de dados.

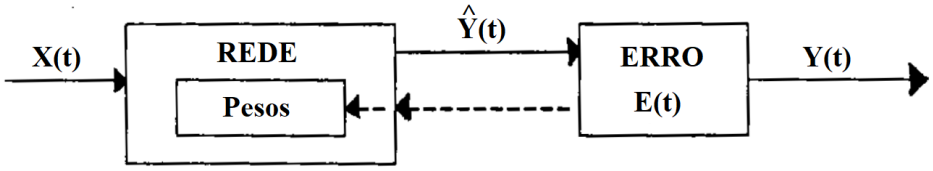
Referente ao processo de treinamento em si, os valores dos pesos de cada neurônio são ajustados gradualmente para se adequarem aos dados inseridos e o algoritmo mais comum de aprendizado é o *backpropagation*, que consiste basicamente no ajuste gradual dos pesos na direção inversa do gradiente da função erro. Dessa forma, ao ajustar os pesos com o intuito de reduzir o erro, esse algoritmo possibilita que a rede tenha mais precisão na identificação de padrões, ou seja, ele permite que a rede se adapte e consiga

ter um melhor desempenho nas tarefas de reconhecimento.

O algoritmo consiste basicamente de três etapas. Na primeira etapa, um sinal de entrada é aplicado ao sistema, sendo propagado pelas camadas e gerando um sinal de saída, o qual apresentará um erro em relação ao sinal de saída desejado. Em seguida, o erro calculado é retropropagado a partir das saídas, verificando a influência de cada uma das sinapses no erro obtido. Por fim, os valores dos pesos de cada uma das sinapses são ajustados de acordo com a influência de cada uma no erro final.

Um esquema do funcionamento desse algoritmo é apresentado na Figura 2.3. Assim, o sinal de entrada X(t) é aplicado na rede neural, gerando o sinal de saída $\hat{Y}(t)$, o qual apresenta um erro E(t) em relação ao sinal de saída esperado Y(t). O sinal pontilhado indica que o erro é retropropagado para a rede, realizando o ajuste nos pesos da mesma.

Figura 2.3 – Algoritmo *backpropagation*



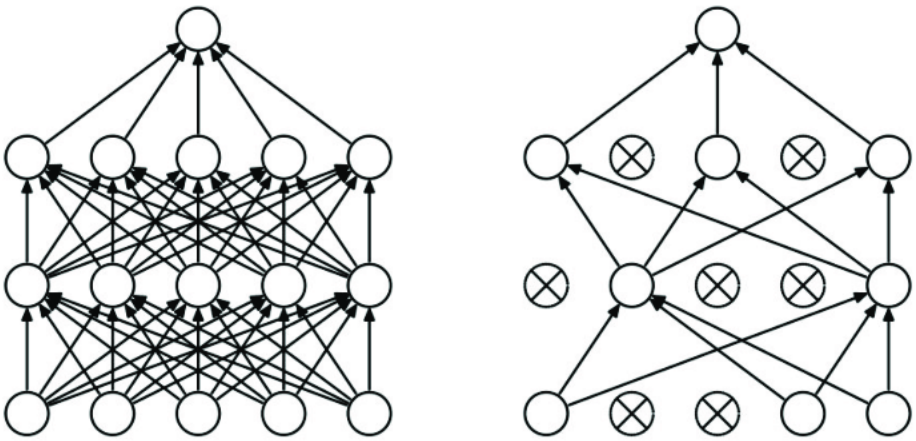
Fonte: Adaptado de Werbos (1990)

Cabe também salientar que a rede neural possui um parâmetro denominado *learning rate*, que é a taxa de aprendizagem da rede. Esse parâmetro influencia em quanto os pesos vão ser ajustados. Maiores taxas de aprendizagem geram ajustes com passos maiores e menores taxas geram ajustes com passos menos significativas.

Geralmente, o processo de treinamento é iniciado com uma taxa de aprendizagem alta e, à medida em que a rede se aproxima de boas soluções, essa taxa é reduzida para que a rede não se distancie muito dessas soluções. Dessa forma, inicia-se o treinamento evitando mínimos locais, mas com maior risco de acabar não convergindo para os pesos de melhor performance, e ao reduzir essa taxa, tem-se um maior risco de se prender a mínimos locais, mas convergindo mais facilmente ao valor ótimo quando se encontra próxima do mínimo global.

Um outro mecanismo que pode ser utilizado para melhorar a performance das redes neurais artificiais é a realização de um *dropout*, que consiste em ignorar uma certa quantidade de neurônios durante a realização do treinamento da rede para evitar que ocorra um *overfitting* dos dados, ou seja, para evitar que a rede simplesmente memorize os dados de entrada inseridos. Uma representação dessa técnica é apresentada na Figura 2.4.

Figura 2.4 – Exemplo de rede neural artificial sem e com *dropout*



Fonte: ResearchGate[1]

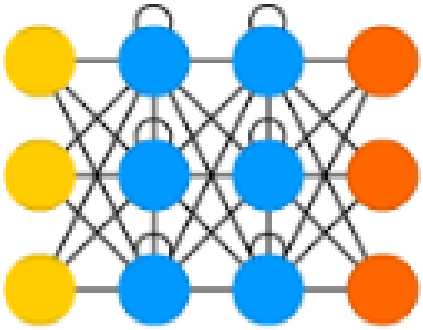
Finalmente, a estrutura de rede neural apresentada até aqui e ilustrada na Figura 2.2 é de uma rede neural *feedforward*, também conhecida como *multilayer perceptron* (MLP), onde as conexões ocorrem apenas entre neurônios de uma camada para a camada seguinte, sem conexões para elementos da mesma camada ou de camadas anteriores. No próximo tópico, será apresentada uma outra estrutura de redes neurais.

2.1.1 Redes Neurais Recorrentes

Ao contrário das redes *feedforward*, as redes neurais recorrentes são estruturas em que os neurônios se conectam livremente, podendo se conectar tanto aos elementos da próxima camada e da camada anterior, permitindo até que um neurônio se conecte a si próprio. Essas conexões formam laços recorrentes, possibilitando que sinais possam se propagar repetidamente seguindo caminhos fechados. Uma representação de rede neural recorrente é apresentada na Figura 2.5, onde as células amarelas representam os sinais de entrada, as laranjas são os sinais de saída e as azuis representam as células recorrentes, que podem se conectar livremente.

[1]Disponível em: https://www.researchgate.net/figure/4-An-illustration-of-the-dropout-mechanism-within-the-proposed-CNN-a-Shows-a_fig27_317277576. Acesso em: 20 abr. 2018.

Figura 2.5 – Modelo de rede neural recorrente



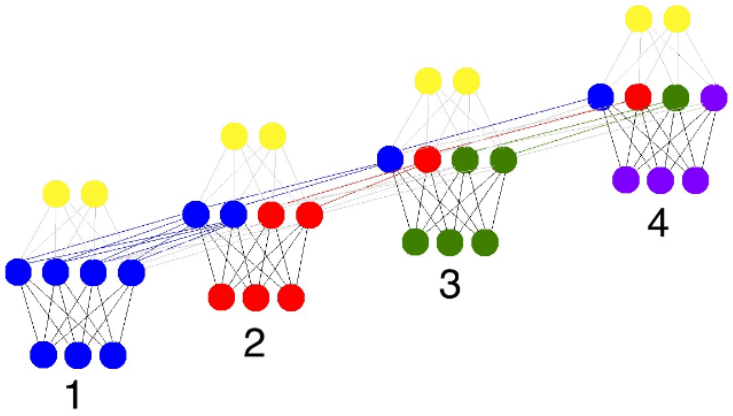
Fonte: (VEEN, 2016)

Dessa forma, por possibilitar que um sinal permaneça muito mais tempo circulando na rede, aumenta-se a chance de serem formadas relações ao longo do tempo, como em sequências dinâmicas. Portanto, redes neurais recorrentes são muito aplicadas nessas sequências dinâmicas, onde os dados futuros apresentam uma certa relação com os anteriores, como é o caso do reconhecimento dinâmico de dígitos.

O método *backpropagation* que foi apresentado para aplicação em redes neurais *feedforward* pode ser estendido para o método de *backpropagation through time*, o qual é utilizado em redes neurais recorrentes. Basicamente, esse método realiza um desdobramento da rede neural ao longo do tempo, empilhando cópias da rede neural em sequência e transformando-a em uma rede neural *feedforward*, na qual é possível utilizar o método de *backpropagation*.

Na Figura 2.6, pode-se observar o desdobramento da rede ao longo de 4 instantes de tempo para aplicação desse método. Assim, as células inferiores, em azul, vermelho, verde e roxo, representam respectivamente as quatro entradas da rede neural ao longo desses instantes de tempo. As células intermediárias representam as camadas ocultas da rede e são apresentadas com combinações das cores citadas para representar que elas dependem tanto da entrada atual quanto das camadas anteriores. As células amarelas representam as saídas em cada um dos quatro instantes de tempo. Dessa forma, com a rede desdobrada ao longo do tempo, é aplicado o método de *backpropagation*, onde os sinais de entradas são propagados por toda a rede, os erros são calculados e retropropagados partindo do último desdobramento da rede e então são realizados os ajustes dos pesos.

Figura 2.6 – Algoritmo *backpropagation through time*



Fonte: (TRASK, 2015)

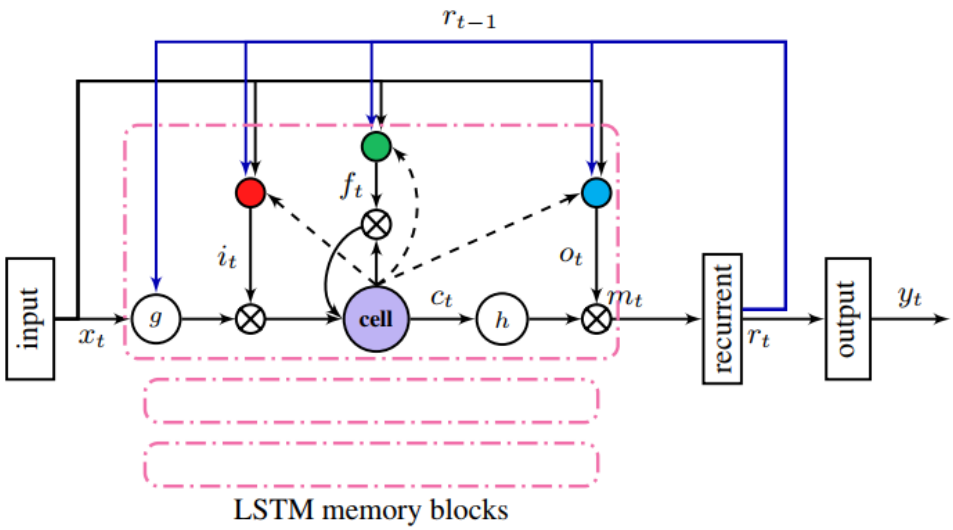
2.1.2 *Long Short-Term Memory*

Uma rede LSTM é uma rede de Memória de Curto e Longo Prazo. Por ser uma especialização de uma rede neural recorrente, sua arquitetura é semelhante à apresentada na Figura 2.5.

O diferencial de uma rede LSTM para uma rede neural recorrente qualquer é que ela possui unidades especiais denominadas blocos de memória, que foram projetadas especialmente para modelar sequências temporais e suas dependências à longo prazo, como afirmam Sak, Senior e Beaufays (2014). Cada bloco é composto por uma célula de memória, um *input gate*, um *forget gate* e um *output gate*.

Na Figura 2.7, é representada uma rede neural LSTM com esse bloco de memória destacado com um contorno pontilhado rosa. A célula de memória é representada por *cell*, os *gates* são denotados por $\otimes$, os vetores de ativação da célula, *input gate, forget gate* e *output gate* são denotados por c_t, i_t, f_t e o_t, respectivamente. Os sinais de entrada e saída denotam-se como x_t e m_t, enquanto as funções de ativação de entrada e saída da célula são representadas por g e h, respectivamente.

Figura 2.7 – Representação de um bloco de memória



Fonte: (SAK; SENIOR; BEAUFAYS, 2014)

As equações a seguir se baseiam no que foi publicado por Olah (2015). Assume-se que todas as variáveis W são matrizes que representam os pesos das conexões recorrentes e as variáveis b são vetores denominados *biases*, que contribuem na adaptação da rede.

O *forget gate* se baseia na entrada atual do sistema e na saída do instante anterior para determinar um valor σ, que define o quanto de informação deve ser esquecida da célula de memória atual. Esse valor é sempre um número entre 0 e 1, no qual 0 representa que toda informação deve ser esquecida e 1 representa que toda informação deve ser mantida.

$$f_t = \sigma(W_f \cdot [m_{t-1}, x_t] + b_f) \tag{2.1}$$

O *input gate* determina que valores serão atualizados para que novas informações possam ser armazenadas na célula de memória e define também quais são os candidatos para esses novos valores, atualizando então o estado da célula.

$$i_t = \sigma(W_i \cdot [m_{t-1}, x_t] + b_i) \tag{2.2}$$

$$c_t^* = tanh(W_c \cdot [m_{t-1}, x_t] + b_c) \tag{2.3}$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot c_t^* \tag{2.4}$$

Finalmente, o *output gate* determina que informações são relevantes para serem dadas como fornecidas como dados de saída em cada instante de tempo.

$$o_t = \sigma(W_o \cdot [m_{t-1}, x_t] + b_o) \tag{2.5}$$

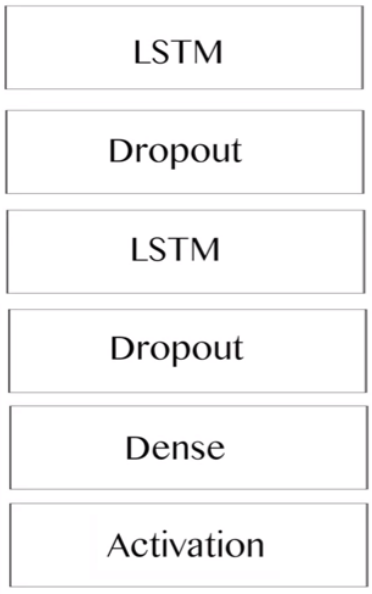
$$m_t = o_t \cdot tanh(c_t) \tag{2.6}$$

2.1.3 Modelos Geradores

Devido à grande aplicabilidade na análise e modelagem de sequências, as redes LSTM, além de serem utilizadas para classificar e identificar sequências, podem ser utilizadas para reconhecer padrões e gerar novas sequências com base nos dados utilizados para treinamento.

Além das possíveis aplicações desses modelos de rede neural em produção de texto e previsões prolongadas de sequências, Raval (2017) apresenta um exemplo de implementação na geração de música e afirma também que a estrutura de rede neural que pode ser observada na Figura 2.8 é mais adequada para esses modelos por permitir uma maior generalização ao inserir duas camadas LSTM e não apenas uma, possibilitando melhores previsões.

Figura 2.8 – Estrutura proposta para um modelo gerador



Fonte: (RAVAL, 2017)

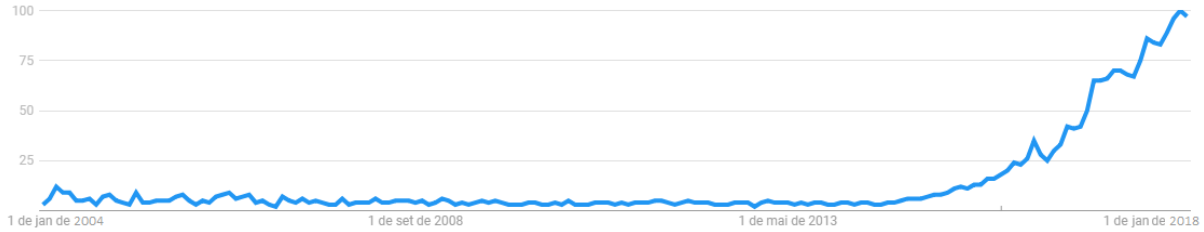
As camadas *Dropout* servem para transformar uma porcentagem dos dados de entrada em informações nulas em cada época do treinamento, para evitar que a rede simplesmente memorize os dados de entrada; a camada *Dense* transforma os dados de treinamento para o formato esperado dos dados de saída e a camada *Activation* utiliza a função *softmax* para fazer com que o vetor de saída se transforme em um vetor de probabilidades, com todos valores entre 0 e 1 e soma igual a 1.

2.2 CONTEXTUALIZAÇÃO E TRABALHOS RELACIONADOS

Até este ponto, foram abordados diversos conceitos referentes a redes neurais e, mais especificamente, a redes LSTM. É importante também analisar a utilização de redes neurais em outros trabalhos e pesquisas.

Dessa forma, as Figuras 2.9 e 2.10 apresentam o índice de pesquisas, de acordo com o Google Trends, pelo termo *"Long Short-Term Memory"* desde 2004 e nos últimos 5 anos, respectivamente.

Figura 2.9 – Índice de pesquisas por *"Long Short-Term Memory"* desde 2004



Fonte: Google Trends[2]

Figura 2.10 – Índice de pesquisas por *"Long Short-Term Memory"* nos últimos 5 anos

Fonte: Google Trends[3]

A partir dessas imagens, pode-se observar a crescente popularidade pelo termo, comprovando que a utilização de redes LSTM é um assunto cada vez mais pesquisado e estudado na atualidade.

Existem diversas variações das células LSTM, entretanto Greff et al. (2017) realizaram um estudo comparando oito diferentes variações da rede LSTM e concluíram que nenhuma dessas variações apresentou melhoras significativas quando comparadas com a rede LSTM padrão. Dessa forma, esse trabalho focará na utilização da rede LSTM mais comum, já apresentada na Seção 2.1.2.

[2]Disponível em: https://trends.google.com.br/trends/explore?date=allq=%2Fm%2F02qmzyq. Acesso em: 26 abr. 2018.

[3]Disponível em: https://trends.google.com.br/trends/explore?date=today%205-yq=%2Fm%2F02qmzyq. Acesso em: 26 abr. 2018.

Na literatura podem ser encontrados outros trabalhos que apresentam aplicações similares às quatro aplicações propostas: reconhecimento de dígitos manuscritos, produção de fragmentos de texto, previsão de índices de poluição do ar e realização de movimentos em um jogo.

Diversos autores realizaram trabalhos de reconhecimento de letras e dígitos manuscritos utilizando as redes neurais LSTM. Dentre eles, Tolosana et al. (2018) exploram a utilização da rede LSTM e de outras redes neurais recorrentes para o reconhecimento de assinaturas manuscritas. Os autores ainda destacam os resultados obtidos, salientando que a rede LSTM foi capaz de obter bons resultados na verificação de assinaturas.

Aliado a isso, Sundermeyer, Ney e Schlüter (2015) realizam uma comparação de uma tarefa de modelagem de linguagem utilizando redes neurais *feedforward*, redes neurais recorrentes e redes LSTM. Por meio desse trabalho, os autores relataram que as redes *feedforward* são superadas pelas redes neurais recorrentes e que estas são superadas pelas redes LSTM, afirmando também que redes com maiores números de camadas apresentam resultados melhores do que as redes mais superficiais.

Kok, Simsek e Özdemir (2017) utilizam uma rede LSTM para prever futuros valores de qualidade de ar em uma cidade inteligente, afirmando que o emprego da rede é efetivo e promissor e que os resultados obtidos pela mesma são melhores do que quando comparados com uma outra metodologia de classificação de dados.

Finalmente, Chellapilla e Fogel (1999) aplicam modelos de redes neurais artificiais em jogos como damas, jogo da velha e o dilema do prisioneiro iterado. Por meio dos experimentos realizados, os autores concluíram que redes neurais podem gerar estratégias muito úteis nos mais diversos jogos.

3 MATERIAIS E MÉTODOS

Neste capítulo, serão apresentadas as metodologias referentes às quatro propostas de implementação de redes neurais artificiais. Sendo assim, cada seção será dedicada à explicação de uma das aplicações já mencionadas.

Todas as redes neurais artificiais LSTM propostas foram implementadas em Python, utilizando a biblioteca Keras, desenvolvida por Chollet et al. (2015) para a programação em alto nível de redes neurais.

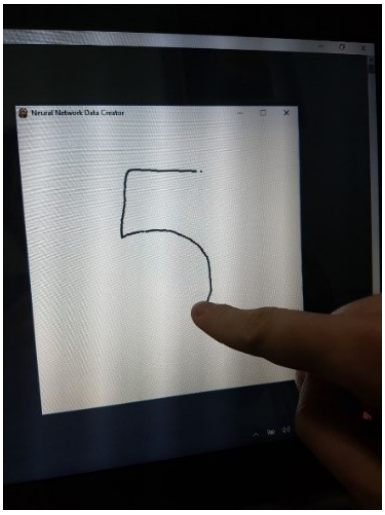
3.1 RECONHECIMENTO E CLASSIFICAÇÃO DE DÍGITOS EM TEMPO REAL

Nesta seção, serão apresentados os pontos principais dos algoritmos implementados na classificação de dígitos para a criação do banco de dados, seu pré-processamento, a criação e treinamento da rede neural, sua aplicação em dados já prontos e sua aplicação em tempo real, explicando algumas das bibliotecas utilizadas, como a Pygame e Pickle, ambas do Python.

3.1.1 Obtenção e processamento dos dados

Inicialmente, para a geração dos dados, foi utilizada a biblioteca Pygame para criar uma interface gráfica consistindo em uma janela de 500x500 pixels na qual deveria ser escrito um dígito por meio de uma tela sensível ao toque. Na Figura 3.1, pode ser observado o desenho de um dígito 5 utilizando a tela touchscreen de um notebook.

Figura 3.1 – Desenho de um dígito utilizando o Pygame e uma tela touchscreen

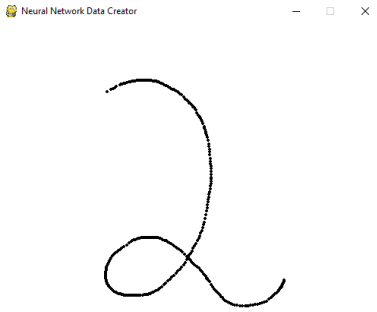


Fonte: Elaborada pelo autor

O método de treinamento da rede escolhido foi o de aprendizagem supervisionada, tendo como entrada informações sobre as posições do mouse ao longo do tempo e os rótulos dos dígitos desenhados. Assim, um código foi implementado para que, à medida em que o dígito fosse sendo escrito, a posição em x e em y que o mouse se encontrava na tela fosse capturada a cada 5ms com o formato (x,y). Ao finalizar a escrita do dígito e apertar a barra de espaço, o programa solicitava que se identificasse qual dígito foi escrito, gravando as posições do mouse ao longo da escrita e a identificação desse dígito em um vetor.

Assim, na Figura 3.2 é apresentada uma imagem da tela do notebook após completar o desenho de um dígito 2. Verifica-se que a linha do desenho não é continua, pois as posições são amostradas e desenhadas a cada 5ms, gerando um conjunto discreto de pontos.

Figura 3.2 – Exemplo de um dígito criado para o banco de dados

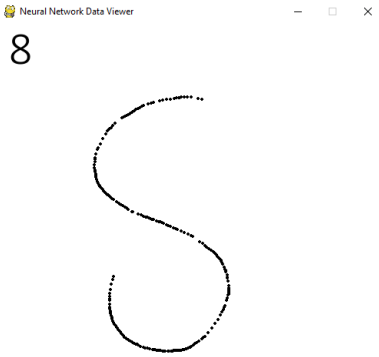


Fonte: Elaborada pelo autor

Dessa forma, utilizou-se a biblioteca Pickle para gravar os dados de todos os dígitos que foram desenhados, permitindo que os dados fossem reutilizados por outras aplicações. Assim, foi construída uma base de dados com 1000 dígitos para treinamento, 100 para validação e 100 para teste, tendo uma distribuição uniforme dos 10 dígitos (100 desenhos de cada um dos dígitos para treinamento e 10 de cada para validação e teste).

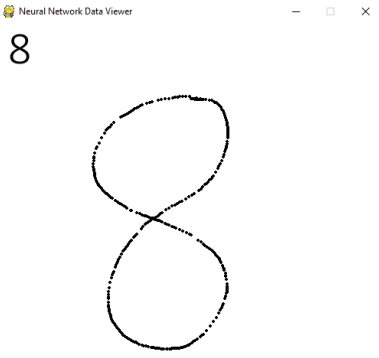
Após a geração dos dados e a construção do banco de dados, para confirmar que os dados estavam sendo gravados de forma coerente e com a identificação correta, foi desenvolvido um algoritmo que desenhava cada um dos dígitos na tela e apresentava o rótulo ou dígito ao qual ele foi associado. Diferentes etapas do desenho de um mesmo dígitos podem ser observadas nas Figuras 3.3 e 3.4.

Figura 3.3 – Início do desenho de um dígito e sua identificação



Fonte: Elaborada pelo autor

Figura 3.4 – Desenho do dígito completo e sua identificação



Fonte: Elaborada pelo autor

Após confirmar que o banco de dados foi gravado adequadamente e os dígitos foram rotulados corretamente, foi desenvolvido um novo algoritmo, o qual é apresentado no Apêndice A, para realizar o pré-processamento dos dados. Dessa forma, partindo do

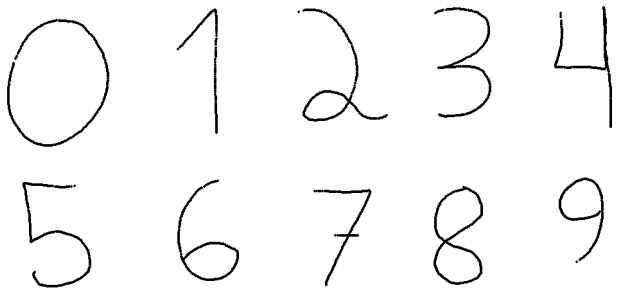
conjunto de posições dos dígitos, criou-se um conjunto de dados que continha as sequências Δx e Δy, representando a variação das posições em x e em y ao longo da escrita dos dígitos. Essas sequências possuíam uma formatação específica que será detalhada posteriormente.

Essa mudança é interessante, uma vez que se espera que a rede reconheça os dígitos independentemente de onde eles forem escritos na tela. Portanto, realmente é mais adequado considerar as variações nas posições do que as posições em si.

Cabe salientar que como a rede não reconhece simplesmente a composição final do dígito e sim a construção dinâmica do mesmo, a ordem em que é realizada a construção do dígito é de extrema importância. Assim, para obter uma rede com resultados mais consistentes, todos os dígitos foram desenhados seguindo o mesmo padrão, o qual é descrito no próximo parágrafo e gerando resultados finais semelhantes aos apresentados na Figura 3.5.

O dígito 0 é desenhado no sentido anti-horário começando do ponto central superior. O 1 é desenhado sem a base, começando com um traço inclinado para cima e para a direita e encerrando com um traço vertical para baixo. O 2 e o 3 são desenhados continuamente, iniciando na parte superior esquerda. O dígito 4 é iniciado com um traço para baixo, um para direita, outro para cima e, sem retirar o dedo da tela, um último traço longo até a parte final do dígito. Os dígitos 5 e 6 são desenhados continuamente começando na parte superior direita. O 7 é desenhado começando com um traço para a direita partindo da parte superior esquerda, um traço inclinado para baixo e para a esquerda e, após retirar o dedo da tela, um traço horizontal no centro do dígito. O 8 é desenhado iniciando no ponto central superior e no sentido anti-horário, prosseguindo a construção da parte inferior no sentido horário e retornando à parte superior no sentido anti-horário. Finalmente, o dígito 9 é desenhado continuamente iniciando do ponto inferior esquerdo.

Figura 3.5 – Representação final de cada um dos dígitos



Fonte: Elaborada pelo autor

Quanto à formatação do banco de dados, era necessário que todas as amostras possuíssem o mesmo comprimento para serem aplicadas no treinamento da rede neural. Portanto, à medida em que eram calculadas as variações das posições no algoritmo,

verificou-se também que o comprimento máximo dos vetores era de 600 intervalos de tempo, que equivalem a 600*5ms=3s.

Assim, o código foi implementado de forma que todos os vetores fossem completados com (0,0) até atingirem um comprimento 600. Isso seria equivalente a afirmar que após o término da construção do dígito, a variação do mouse fosse 0 em x e 0 em y, ou seja, que o mouse permaneceu parado.

Além disso, essa adaptação fez com que muitos vetores ficassem com diversos valores nulos nas últimas posições. Sabendo que a rede LSTM valoriza as posições finais da sequência, optou-se por inverter cada um dos vetores, fazendo com que as variações nulas fossem alocadas nos primeiros vetores e permitindo que os valores mais significativos fossem mais bem aproveitados pela rede. Por fim, utilizou-se novamente a biblioteca Pickle para salvar o banco de dados já processado.

3.1.2 Treinamento da rede neural

Com o banco de dados pronto e pré-processado, foi possível prosseguir para a construção da rede e o treinamento da mesma. Assim, o código utilizado para isso é apresentado no Apêndice B.

Inicialmente, são carregados os dados dos dígitos de treinamento e de validação que já foram pré-processados, separando-os entre os dados das variações de posição e os rótulos de quais dígitos são representados nos dados. Esses rótulos são convertidos para o formato de *one-hot encoding*, fazendo com que um rótulo 0 seja convertido para [1,0,0,0,0,0,0,0,0,0] e um rótulo 5 para [0,0,0,0,0,1,0,0,0,0], por exemplo. Essa escrita é conveniente para calcular o erro, conforme explicado na Seção 3.1.

Então, define-se que o nosso modelo será composto por uma pilha linear ou sequência de camadas e, para se ter um algoritmo de *Deep Learning*, são adicionadas três camadas da rede neural recorrente LSTM. Cada camada é composta por 32 neurônios e assume que o formato da entrada é de vetores com 600 de comprimento e 2 dimensões (Δx e Δy).

Finalmente, são acrescentadas uma camada densa e uma camada de ativação. A camada densa define o formato da saída como um vetor de comprimento 10, enquanto a camada de ativação basicamente utiliza a função *softmax* para fazer com que cada elemento desse vetor corresponda às probabilidades de o dígito desenhado ser identificado como cada um dos dígitos de 0 a 9.

Para definir a estrutura da rede, foram testadas diversas estruturas com um número diferente camadas da rede LSTM e com diferentes quantidades de neurônios por camada. Optou-se pela estrutura aqui apresentada por ter sido a que apresentou melhores resultados, embora consumisse um tempo considerável de treinamento.

Quanto aos detalhes técnicos utilizados para o treinamento, o erro adotado foi o *categorical crossentropy*, já abordado na Seção 2.1, e a métrica utilizada foi a precisão, ou seja, definiu-se que o desempenho da rede seria avaliado de acordo com o percentual de vezes em que o número previsto pela rede é realmente o correto.

Com o modelo pronto, foi possível seguir para o treinamento da rede para o reconhecimento de dígitos, que é o foco deste trabalho. Assim, a rede é treinada para reconhecer os dígitos por meio dos dados e rótulos de treinamento e o seu desempenho é verificado pela precisão e pelo erro tanto nos dados de treinamento quanto nos de validação.

Ainda, são utilizadas três funções para observar e fazer alterações no processo de treinamento. Uma delas é aplicada para gravar o modelo que apresentar menor erro de validação ao final de cada época, outra reduz a taxa de aprendizagem se a rede não apresentar uma redução no erro de validação em 2 épocas seguidas e a última encerra o treinamento antes do limite pré-definido de 100 épocas se ocorrerem 6 épocas seguidas sem uma redução no erro de validação. Finalmente, após o treinamento da rede, a mesma é salva em um arquivo para uso futuro em outros dados e aplicações.

O algoritmo apresentado no Apêndice C foi desenvolvido com o intuito de carregar o modelo de uma rede que já foi treinada, testá-la em um novo conjunto de dados e verificar a precisão da mesma.

De forma análoga ao algoritmo anterior, inicialmente são carregados os dados de teste com seus rótulos e estes são transformados para o formato de *one-hot encoding*. É também carregado o modelo de rede neural artificial que já foi implementado e treinado anteriormente, conservando a sua estrutura e todas as matrizes com seus pesos atualizados pelo treinamento.

Utilizou-se o comando *model.predict()* para verificar a precisão do modelo, contando para quantos dígitos o resultado previsto pela rede era equivalente ao rótulo dado ao dígito. Com isso, além da precisão da rede nesses dados de teste, era mostrado na tela também os dígitos que a rede não reconheceu corretamente.

Finalmente, por meio de uma implementação semelhante à apresentada nesse código, adaptou-se o algoritmo que foi utilizado para desenhar os dígitos e gravar o banco de dados para que a rede funcionasse em tempo real. Com isso, o código desenvolvido basicamente captura os dados utilizando a mesma tela inicial do Pygame, realiza o processamento dos dados ao longo do programa e, à medida em que os dígitos são escritos, são utilizadas partes desse último algoritmo para que a rede faça o reconhecimento e para que seja apresentado na tela o número identificado por ela.

Exemplos da aplicação desse código e de algumas variações realizadas, como a possibilidade de escrever e reconhecer até 4 dígitos simultaneamente em uma única tela ou de permitir que sejam escritos vários dígitos em uma tela maior e que a rede reconheça um por vez, à medida que são escritos, serão apresentados com maior detalhe no fim da

próxima seção.

3.2 PRODUÇÃO DE FRAGMENTOS DE TEXTO

Nesta seção, serão apresentados alguns dos algoritmos e procedimentos para a implementação de uma rede neural artificial para a produção de fragmentos de texto.

3.2.1 Obtenção e processamento dos dados

Para realizar o treinamento da rede, era necessário um banco de dados com inúmeros textos ou frases. Dessa forma, por ser uma obra extensa e de domínio público, optou-se por utilizar a Bíblia Sagrada como base de dados.

Para isso, por ser um formato que pode ser facilmente processado em Python, foi realizado o *download* da mesma no formato de um arquivo de texto[1]. Na Figura 3.6, é apresentado um trecho do texto antes de qualquer processamento.

Figura 3.6 – Trecho do arquivo de texto utilizado

```
»I CORINTIOS [13]
1 Ainda que eu falasse as línguas dos homens e dos anjos, e não tivesse
amor, seria como o metal que soa ou como o címbalo que retine.
2 E ainda que tivesse o dom de profecia, e conhecesse todos os mistérios
e toda a ciência, e ainda que tivesse toda fé, de maneira tal que
transportasse os montes, e não tivesse amor, nada seria.
3 E ainda que distribuísse todos os meus bens para sustento dos pobres, e
ainda que entregasse o meu corpo para ser queimado, e não tivesse amor,
nada disso me aproveitaria.
4 O amor é sofredor, é benigno; o amor não é invejoso; o amor não se
vangloria, não se ensoberbece,
5 não se porta inconvenientemente, não busca os seus próprios interesses,
não se irrita, não suspeita mal;
6 não se regozija com a injustiça, mas se regozija com a verdade;
7 tudo sofre, tudo crê, tudo espera, tudo suporta.
8 O amor jamais acaba; mas havendo profecias, serão aniquiladas; havendo
línguas, cessarão; havendo ciência, desaparecerá;
9 porque, em parte conhecemos, e em parte profetizamos;
10 mas, quando vier o que é perfeito, então o que é em parte será
aniquilado.
11 Quando eu era menino, pensava como menino; mas, logo que cheguei a ser
homem, acabei com as coisas de menino.
12 Porque agora vemos como por espelho, em enigma, mas então veremos face
a face; agora conheço em parte, mas então conhecerei plenamente, como
também sou plenamente conhecido.
13 Agora, pois, permanecem a fé, a esperança, o amor, estes três; mas o
maior destes é o amor.
```

Fonte: Elaborada pelo autor

Como pode ser observado, o texto original continha indicações de capítulos e versículos da Bíblia Sagrada representados por números ao final do nome do capítulo ou no

[1] Disponível em: https://sojesuscristosalva.wordpress.com/2011/07/27/biblia-sagrada-em-txt/. Acesso em: 23 jan. 2018.

início de cada nova linha. Porém, com o intuito de obter resultados que não contenham numerações desnecessárias, optou-se por remover tais númerações. Dessa forma, o mesmo trecho pode ser observado na Figura 3.7 após as adaptações.

Figura 3.7 – Trecho do arquivo de texto utilizado após remover as numerações

```
»I CORINTIOS
 Ainda que eu falasse as línguas dos homens e dos anjos, e não tivesse
amor, seria como o metal que soa ou como o címbalo que retine.
 E ainda que tivesse o dom de profecia, e conhecesse todos os mistérios e
toda a ciência, e ainda que tivesse toda fé, de maneira tal que
transportasse os montes, e não tivesse amor, nada seria.
 E ainda que distribuísse todos os meus bens para sustento dos pobres, e
ainda que entregasse o meu corpo para ser queimado, e não tivesse amor,
nada disso me aproveitaria.
 O amor é sofredor, é benigno; o amor não é invejoso; o amor não se
vangloria, não se ensoberbece,
 não se porta inconvenientemente, não busca os seus próprios interesses,
não se irrita, não suspeita mal;
 não se regozija com a injustiça, mas se regozija com a verdade;
 tudo sofre, tudo crê, tudo espera, tudo suporta.
 O amor jamais acaba; mas havendo profecias, serão aniquiladas; havendo
línguas, cessarão; havendo ciência, desaparecerá;
 porque, em parte conhecemos, e em parte profetizamos;
 mas, quando vier o que é perfeito, então o que é em parte será
aniquilado.
 Quando eu era menino, pensava como menino; mas, logo que cheguei a ser
homem, acabei com as coisas de menino.
 Porque agora vemos como por espelho, em enigma, mas então veremos face a
face; agora conheço em parte, mas então conhecerei plenamente, como
também sou plenamente conhecido.
 Agora, pois, permanecem a fé, a esperança, o amor, estes três; mas o
maior destes é o amor.
```

Fonte: Elaborada pelo autor

Finalmente, para facilitar o treinamento e fazer com que a rede não diferenciasse letras maiúsculas e minúsculas, o texto foi todo convertido para conter apenas letras minúsculas. O código utilizado para realizar essa operação e todo o treinamento foi adaptado de um arquivo disponibilizado no Github[2] e é apresentado no Apêndice D.

Assim, o texto utilizado para treinamento da rede contava com um total de 3906470 caracteres, sendo representados por 56 caracteres distintos. Nesse ponto, cabe salientar que a escrita realizada por meio desse algoritmo é realizada caractere a caractere.

Para isso, os sinais de entradas da rede foram definidos como sequências de 40 caracteres do texto e os sinais de saída foram definidos como o caractere posterior à sequência correspondente no texto. Além disso, as sequências foram definidas a partir do início do texto tomando 10 caracteres como intervalo entre elas. Portanto, o primeiro sinal de entrada corresponde aos 40 primeiros caracteres do texto e o primeiro sinal de saída corresponde ao 41º caractere, o segundo sinal de entrada corresponde aos caracteres 11 a 50 e o segundo sinal de saída corresponde ao 51º caractere, e assim sucessivamente.

Para compreender melhor o processo de transformação do texto em sinais de entrada e saída para o treinamento, pode-se considerar como exemplo o texto "redes neurais artificiais" e dividi-lo em sequências com comprimento fixo de 7 caracteres e com interva-

[2]Disponível em: https://archive.ics.uci.edu/ml/datasets/Beijing+PM2.5+Data. Acesso em: 23 fev. 2018.

los de 3 caracteres entre elas. Assim, os dados de entrada e de saída seriam como os apresentados na Tabela 3.1.

Tabela 3.1 – Exemplo de transformação de um trecho de texto em dados de entrada e saída.

Sequência de entrada	Saída esperada
"redes n"	"e"
"es neur"	"a"
"neurais"	” ”
"rais ar"	"t"
"s artif"	"i"
"rtifici"	"a"

Fonte: Elaborada pelo autor

Por meio dessa transformação, o nosso texto de 3906470 caracteres foi transformado em 390643 sequências de 40 caracteres cada. Embora a notação utilizada até o momento seja de mais fácil compreensão, para o treinamento da rede neural cada caractere foi convertido para o formato de *one-hot encoding*. Como o texto continha 56 caracteres distintos, cada caractere foi transformado em um vetor de comprimento 56, com todos elementos sendo iguais a 0, exceto o elemento correspondente ao caractere desejado, que possuía valor igual a 1.

Dessa forma, com cada um dos caracteres de entrada e de entrada sendo transformados para o formato de *one-hot encoding*, cada sequência de 40 caracteres foi convertida para uma matriz de 40 colunas e 56 linhas, onde cada coluna corresponde a um dos caracteres descritos. Com isso, o nosso banco de dados para essa aplicação foi composto por essas 390643 matrizes 40x56 e os 390643 vetores de comprimento 56 correspondentes ao caractere de saída esperados de cada uma delas.

3.2.2 Treinamento e aplicação da rede neural

Após realizar a construção de todo o banco de dados, foi utilizado o modelo proposto por Raval (2017) e apresentado na Seção 2.1.3.

Assim, o modelo de rede neural foi definido como uma sequência linear de camadas, contendo duas camadas LSTM e duas camadas *Dropout*, alternadamente, produzindo assim um algoritmo de *Deep Learning*. Cada camada *Dropout* é responsável por anular 20% dos coeficientes ao longo do treinamento, contribuindo para evitar que a rede simplesmente decore os dados de entrada, e cada camada LSTM é composta por 128 neurônios e recebe vetores no formato de matrizes 40x56, que equivale às sequências de 40 caracteres que foram transformadas inicialmente.

Por fim, são adicionadas uma camada densa, que define os vetores de saída como vetores de comprimento 56, e uma camada de ativação. Essa camada final utiliza a função *softmax* para apresentar um vetor que corresponde às probabilidades de o próximo algarismo esperado ser cada um dos 56 caracteres possíveis de serem escritos.

Assim como na utilização para o reconhecimento de dígitos, o erro adotado foi o *categorical crossentropy* e a métrica utilizada foi a precisão, definindo que o desempenho da rede é avaliado de acordo com o percentual de vezes em que o caractere previsto pela rede é realmente o correto.

Com a definição do modelo, prosseguiu-se para o treinamento da rede com o banco de dados adequado. Após o treinamento, introduziu-se o código para a produção de texto em si. Assim, partindo de uma das sequências de 40 caracteres utilizadas para o treinamento, a rede prevê o próximo caractere buscando gerar alguma coerência. Em seguida, são utilizados os 39 últimos caracteres da sequência original e o novo caractere previsto para realizar a previsão de um próximo caractere.

Dessa forma, o processo de prever um novo caractere e utilizar os últimos 40 caracteres do texto produzido para realizar uma nova previsão pode ser repetido quantas vezes for desejado. Um exemplo do funcionamento desse algoritmo pode ser observado na Tabela 3.2, no qual a sequência de entrada seria o fragmento de 40 caracteres "redes neurais artificiais long short-ter" e o texto gerado seria "redes neurais artificiais *long short-term memory*".

Tabela 3.2 – Exemplo de como o algoritmo produz um fragmento de texto.

Sequência de entrada	Saída gerada
"redes neurais artificiais long short-ter"	"m"
"edes neurais artificiais long short-term"	” ”
"des neurais artificiais long short-term "	"m"
"es neurais artificiais long short-term m"	"e"
"s neurais artificiais long short-term me"	"m"
"neurais artificiais long short-term mem"	"o"
"neurais artificiais long short-term memo"	"r"
"eurais artificiais long short-term memor"	"y"

Fonte: Elaborada pelo autor

O exemplo aqui apresentado é apenas para ilustrar como é realizada a produção de texto por meio desse algoritmo, porém os resultados realmente obtidos a partir do treinamento da rede serão apresentados na Seção 4 e o algoritmo utilizado é apresentado no Apêndice E.

3.3 PREVISÕES DE NÍVEIS DE POLUIÇÃO

Nesta seção, serão abordados os métodos utilizados para a aplicação de uma rede neural artificial na previsão de indíces de poluição.

3.3.1 Obtenção e processamento dos dados

Para esta aplicação, foi utilizado o conjunto de dados Beijing PM2.5[3] Esse conjunto se refere a medições realizadas a cada hora na Embaixada dos Estados Unidos em Pequim no período de 1º de janeiro de 2010 a 31 de dezembro de 2014, totalizando 5 anos de dados. Assim, foram analisados um total de $365 \times 5 = 1825$ dias, que correspondem a um total de $1825 \times 24 = 43800$ horas.

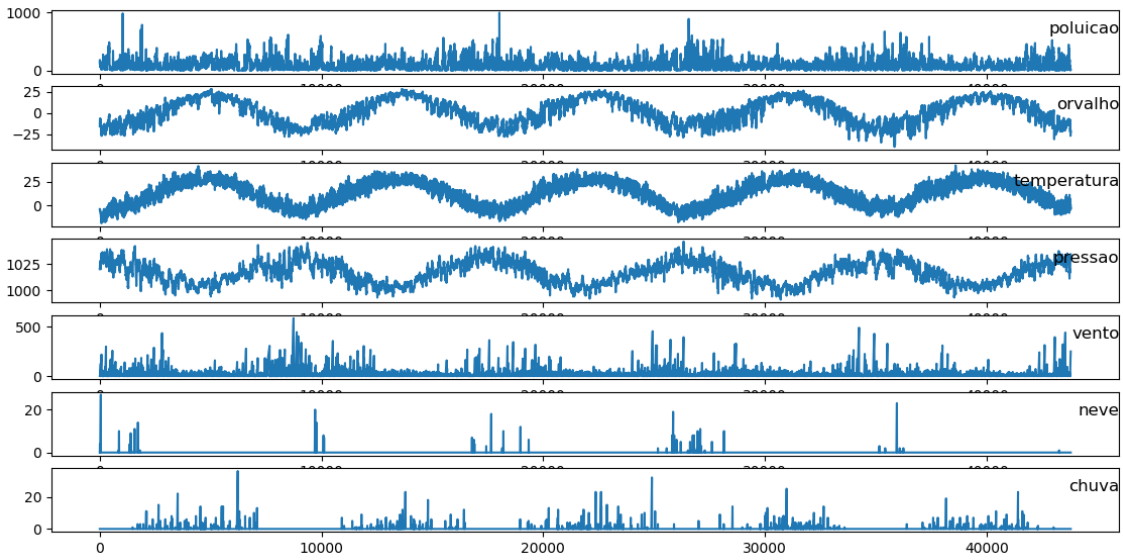
Dessa forma, o nosso conjunto de dados contém informações de 43800 horas, onde cada informação é composta pela data e horário da medição, pela concentração de material particulado (µg/m^3), pelo ponto de orvalho (ºF), temperatura (ºF), pressão (hPa), velocidade (m/s) e direção do vento e horas acumuladas de neve e chuva.

A medição do nível de poluição ocorre por meio da concentração de material particulado PM2.5, que refere-se ao conjunto de partículas no ar com menos de 2.5 micrômetros de diâmetro. Além disso, por ser uma grandeza mais específica, cabe salientar que o ponto de orvalho refere-se à temperatura na qual o vapor de água presente no ar começa a se condensar na forma de pequenas gotas.

Para aplicação na rede neural, foram utilizadas as informações sobre concentração de material particulado PM2.5, ponto de orvalho, temperatura, pressão, velocidade do vento, horas acumuladas de neve e de chuva. Na Figura 3.8 são apresentadas as variações dessas medidas ao longo dos 5 anos de dados coletados.

[3]Disponível em: https://archive.ics.uci.edu/ml/datasets/Beijing+PM2.5+Data. Acesso em: 12 nov. 2017.

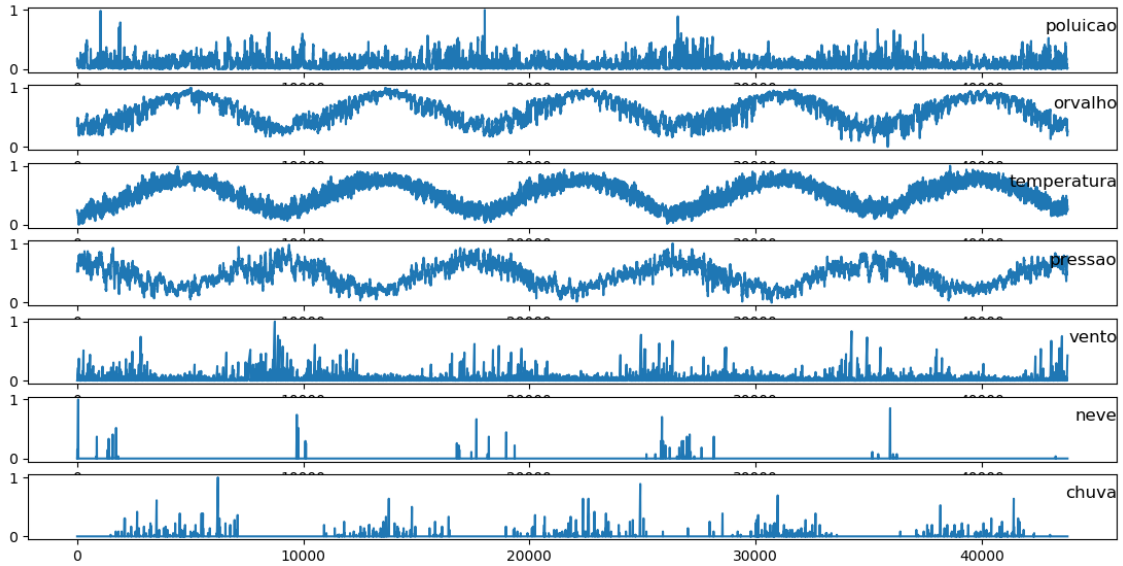
Figura 3.8 – Apresentação do banco de dados utilizado



Fonte: Adaptado de Brownlee (2016)

Para aplicação em redes neurais, é interessante que os valores sejam normalizados, portanto cada uma das 7 características foi ajustada para corresponder a uma escala de 0 a 1 de maneira linear. Na Figura 3.9 pode-se observar que as variações de cada característica são preservadas, alterando apenas os limites de cada uma delas para uma escala de 0 a 1.

Figura 3.9 – Apresentação dos dados normalizados



Fonte: Elaborada pelo autor

Assim, com os dados já normalizados, prosseguiu-se de maneira extremamente análoga à seção anterior, dividindo essa sequência de 43800 horas em diversas sequên-

cias menores para compor os sinais de entrada. Dessa forma, os sinais de entrada para o treinamento foram definidos como sequências de informações relativas a 72 horas consecutivas (ou 3 dias) e os sinais de saída correspondiam às informações na hora subsequente.

Nesse caso, foram utilizadas todas as sequências possíveis. Portanto, a primeira sequência de entrada corresponde às informações nas primeiras 72 horas e a primeira saída corresponde às informações na 73ª hora, a segunda sequência de entrada corresponde às informações da 2ª até a 73ª hora e a segunda saída corresponde às informações da 74ª hora e assim sucessivamente.

Desse modo, como haviam 7 informações em cada hora, o banco de dados utilizado foi transformado em um conjunto de dados de entrada compostos por 43728 matrizes no formato 72x7 e de 43728 vetores de comprimento 7 compondo os dados de saída.

3.3.2 Treinamento e aplicação da rede neural

Por se tratar de um modelo para geração de novas previsões, utilizou-se novamente o modelo apresentado na Seção 2.1.3. O código utilizado para o treinamento desta rede neural pode ser observado no Apêndice F.

Dessa forma, definiu-se o modelo de rede neural como uma sequência de duas camadas LSTM com 256 neurônios cada alternadas com duas camadas *Dropout* que anulavam 20% dos coeficientes durante o treinamento. O formato da entrada era de matrizes 72x7 e as saídas eram vetores de comprimento 7, correspondendo às sequências de informações de 72 horas como entradas e as informações na hora seguinte como saídas correspondentes.

Após isso, foi adicionada uma camada densa para definir a saída do treinamento como um vetor de comprimento 7 e uma camada de ativação. Como a saída nessa aplicação corresponde a informações sobre 7 aspectos climáticos distintos e não a probabilidades de uma única previsão, não foi utilizada a função *softmax*. Assim, a função de ativação simplesmente preserva os valores calculados pelas camadas anteriores.

Diferentemente das aplicações anteriores, o erro adotado foi o erro absoluto médio, definindo que a rede é avaliada de acordo com quão próximos os valores previstos pela rede estão dos valores corretos de cada índice. Tendo o modelo bem definido, foi realizado o treinamento da rede com os dados já processados.

Analogamente à aplicação para a produção de texto, após o término do treinamento, a rede neural foi utilizada para previsão dos índices de poluição com o algoritmo apresentado no Apêndice G. Dessa forma, tendo como base uma das sequências de 72 horas utilizadas no treinamento, a rede prevê inicialmente os índices na hora seguinte. Então, utilizam-se as últimas 71 horas da sequência inicial juntamente com as últimas informa-

ções previstas para realizar a previsão da hora seguinte, e assim sucessivamente pode-se gerar uma sequência tão longa quanto se queira.

3.4 REALIZAÇÃO DE MOVIMENTOS NO JOGO PONG

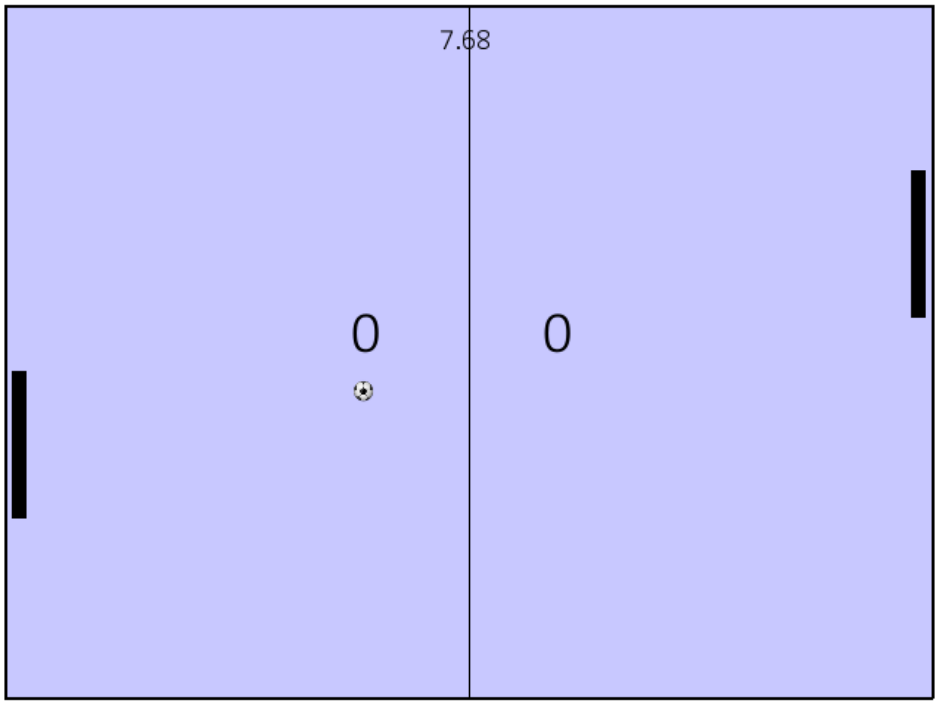
Os mecanismos desenvolvidos para a aplicação de redes neurais artificiais na realização de movimentos no jogo Pong serão apresentados nesta seção.

3.4.1 Obtenção e processamento dos dados

Inicialmente, o jogo Pong utilizado para esta aplicação foi adaptado de um algoritmo desenvolvido ao longo da disciplina de Estrutura de Dados para Automação, componente curricular obrigatório para a conclusão do curso de Engenharia de Controle e Automação na UFSM. A interface gráfica foi construída utilizando a biblioteca Pygame, do Python, e o algoritmo utilizado para gravar os dados é apresentado no Apêndice H.

O jogo é para 2 jogadores e considera-se um ponto quando a bola atinge a linha de fundo do campo adversário. Além disso, quando um dos jogadores faz um ponto, a bola e as barras retornam às posições centrais e a bola parte em direção ao campo do jogador que pontuou. A Figura 3.10 apresenta a tela de jogo, onde podem ser observados o placar dos jogadores, o tempo de jogo, o contorno da quadra, a bola e as duas barras controladas pelos jogadores.

Figura 3.10 – Tela do jogo Pong



Fonte: Elaborada pelo autor

Como o monitor possui uma taxa de atualização da tela de 60 Hertz, o jogo foi programado de modo que as posições, velocidades e a tela em si fossem atualizadas a cada $\frac{1}{60}$s. Dessa forma, cada segundo continha informações referentes às 60 atualiações de posições no jogo.

Assim, decidiu-se que o banco de dados seria composto pelas informações de posições e velocidades verticais e horizontais da bola e de posições e velocidades verticais de cada uma das duas barras (nota-se que cada barra se movimenta apenas na vertical). Ou seja, para cada instante analisado, as informações eram compostas por 8 elementos distintos.

Para a construção do banco de dados, o autor deste trabalho realizou a simulação do jogo por 20 minutos, ininterruptamente, e gravou 72000 informações referentes a esse intervalo, uma vez que eram gravadas 60 informações em cada um dos $20 \times 60 = 1200$s de jogo.

Analogamente à aplicação anterior de previsão de índices de poluição, cada um dos elementos foi convertido para valores de 0 a 1 para auxiliar o treinamento da rede neural. Por fim, os dados de entradas foram definidos como sequências de informações de 2 segundos, ou seja, a 120 informações, e os dados de saída foram definidos como a informação referente ao instante posterior a cada sequência.

Foram utilizadas todas as sequências possíveis para teste. Dessa forma, a primeira sequência de entrada refere-se às primeiras 120 informações e a primeira saída refere-se

à 121ª, enquanto a segunda sequência de entrada se refere às informações do instante 2 ao 121 e a segunda saída se refere à 122ª informação, e assim sucessivamente.

Cabe lembrar que cada uma dessas informações acima citadas são compostas por 8 elementos referentes às posições e velocidades da bola e das barras ao longo do jogo. Portanto, as entradas foram compostas por 71880 matrizes 120x8 e os dados de saída foram compostos por 71880 vetores de comprimento 8.

3.4.2 Treinamento e aplicação da rede neural

A rede utilizada nesta aplicação é semelhante a que foi utilizada na aplicação anterior, contendo duas camadas LSTM com 128 neurônios cada intercaladas com duas camadas *Dropout* que anulam 20% dos coeficientes durante o treinamento e seguidas de uma camada densa e uma camada de ativação. O código utilizado no treinamento da rede neural pode ser encontrado no Apêndice I.

Novamente, não se utilizou a função *softmax* pois os dados de saída não correspondiam a probabilidades e sim a elementos distintos e o erro utilizado foi o erro absoluto médio. Assim, a diferença entre a rede aqui implementada e a anterior reside simplesmente no fato de que as entradas nesse caso são matrizes 120x8 e as saídas são vetores de comprimento 8.

Salientando ainda mais a versatilidade de utilização das redes neurais, sem grandes alterações a rede neural foi utilizada para a previsão de movimentos e informações do jogo Pong após o treinamento do mesmo. Assim, partindo de uma sequência de 2s de movimentos no jogo, a rede prevê os movimentos a serem realizados no instante seguinte e assim sucessivamente de acordo com o algoritmo apresentado no Apêndice J.

Cabe ressaltar que embora a rede realize a previsão de todas as posições e velocidades da bola e das barras, o algoritmo utiliza apenas a velocidade prevista para ambas as barras como informações de entrada. Em outras palavras, considera-se que a rede neural seja responsável por controlar apenas a movimentação das barras, como ocorre em um jogo comum de Pong. Por fim, salienta-se que a movimentação neste caso admite qualquer velocidade entre a velocidade máxima para cima e a velocidade máxima para baixo.

4 RESULTADOS

Este capítulo será dividido em 4 seções, cada uma delas correspondendo a uma das aplicações desenvolvidas neste trabalho.

4.1 RECONHECIMENTO E CLASSIFICAÇÃO DE DÍGITOS EM TEMPO REAL

Nesta seção, serão apresentados os principais resultados obtidos na aplicação no reconhecimento e classificação de dígitos, considerando desde o treinamento da rede até a aplicação da rede no reconhecimento de dados pré-amostrados e o reconhecimento de dígitos sendo aplicado em tempo real e com algumas variações do método.

Para o treinamento da rede, foi utilizado o código apresentado no Apêndice B. O treinamento durou um total de 11 horas, durando cerca de 16 minutos em cada uma das 42 épocas. Os dados de precisão e erro nos dados de treinamento e validação de cada época podem ser observados na Tabela 4.1.

Tabela 4.1 – Dados do treinamento da primeira rede.

Época	Erro no treinamento	Precisão no treinamento (%)	Erro na validação	Precisão na validação (%)
1	1.7349	43.2	1.3078	63
2	1.0899	65.2	0.9483	71
3	0.7937	73.4	0.7021	75
4	0.6147	77	0.5945	76
5	0.4825	83.1	0.4774	87
6	0.3621	87.6	0.3154	87
7	0.2750	88.7	0.2994	87
8	0.2324	90.1	0.2100	90
9	0.1583	95.3	0.2900	87
10	0.1620	95.1	0.1077	99
11	0.0636	99.3	0.2235	91
12	0.1256	95.9	0.0396	99
13	0.0450	99.2	0.0504	98
14	0.0378	99.5	0.0508	99
15	0.0146	99.9	0.0682	98
16	0.0997	99.7	0.0294	99
17	0.0111	99.9	0.0091	100
18	0.0442	98.9	0.0920	98
19	0.0949	97.5	0.0121	100
20	0.0086	99.9	0.0096	100
21	0.0308	99.4	0.0865	97
22	0.0143	99.7	0.0049	100
23	0.0199	99.7	0.5616	88
24	0.1655	93.9	0.0110	100
25	0.0063	99.9	0.0102	100
26	0.0032	100	0.0056	100
27	0.0221	99.5	0.0017	100
28	0.0017	100	0.0016	100
29	0.0172	99.8	0.0042	99
30	0.0287	99.2	0.0050	100
31	0.0009	100	0.0010	100
32	0.0005	100	0.0011	100
33	0.0003	100	0.0186	99
34	0.0667	97.7	0.0049	100
35	0.0014	100	0.0032	100
36	0.0004	100	0.0007	100
37	0.0770	99	0.1067	98
38	0.0156	99.8	0.0368	98
39	0.0517	99.3	0.6653	87
40	0.0464	98.7	0.0503	99
41	0.0037	99.9	0.0360	99
42	0.0174	99.7	0.0515	99

Fonte: Elaborada pelo autor

Como explicado na Seção 3.1, a rede gravou tanto o modelo que obteve o menor erro nos dados de validação quanto o modelo obtido após o fim do treinamento. Assim, foi salvo o modelo obtido após 36 épocas de treinamento, que será denominado “Modelo 1”, e o modelo obtido após as 42 épocas de treinamento, que será denominado “Modelo 2”.

Dessa forma, utilizou-se um código semelhante ao apresentado no Apêndice C para verificar a precisão de ambos os modelos nos dados de treinamento, validação e teste. Na Tabela 4.2 são apresentados o índice de acertos de cada um dos modelos.

Tabela 4.2 – Comparação entre os modelos obtidos.

Conjunto de dados	Precisão Modelo 1	Precisão Modelo 2
Treinamento (1000 amostras)	99,9% (999 acertos)	100% (1000 acertos)
Validação (100 amostras)	100% (100 acertos)	99% (99 acertos)
Teste (100 amostras)	98% (98 acertos)	89% (89 acertos)
Total (1200 amostras)	99.75% (1197 acertos)	99% (1188 acertos)

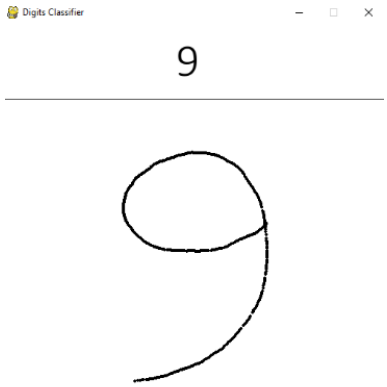
Fonte: Elaborada pelo autor

Verificou-se que ambos os modelos da rede conseguiram reconhecer muito satisfatoriamente os dígitos, identificando no mínimo 89% dos dados de teste aplicados. Além disso, ao considerar os dados de treinamento e validação, a precisão sobe para 99%. Por haver demonstrado uma precisão maior nos dados amostrados, optou-se por utilizar o Modelo 1, o qual será denominado agora simplesmente como “modelo” ou “rede”, para continuar o projeto de reconhecimento de dígitos.

Considerando o bom desempenho dessa rede, foi possível aplicá-la para realizar o reconhecimento de dígitos em tempo real. Na Figura 4.1, é apresentado o primeiro programa desenvolvido com esse propósito, o qual permite que somente um dígito seja escrito na tela e reconhecido por vez.

Esse primeiro código de funcionamento em tempo real foi implementado a partir de uma adaptação do código utilizado para gravar o banco de dados, realizando o processamento dos mesmos e aplicando em um código semelhante ao apresentado no Apêndice C. No caso, assim que o dígito é concluído, apresenta-se o dígito que foi reconhecido pela rede no centro da parte superior da tela, a qual possui dimensões de 500x600 pixels. Para a inserção de um novo dígito, basta pressionar a barra de espaço.

Figura 4.1 – Aplicação inicial do reconhecimento de dígitos em tempo real

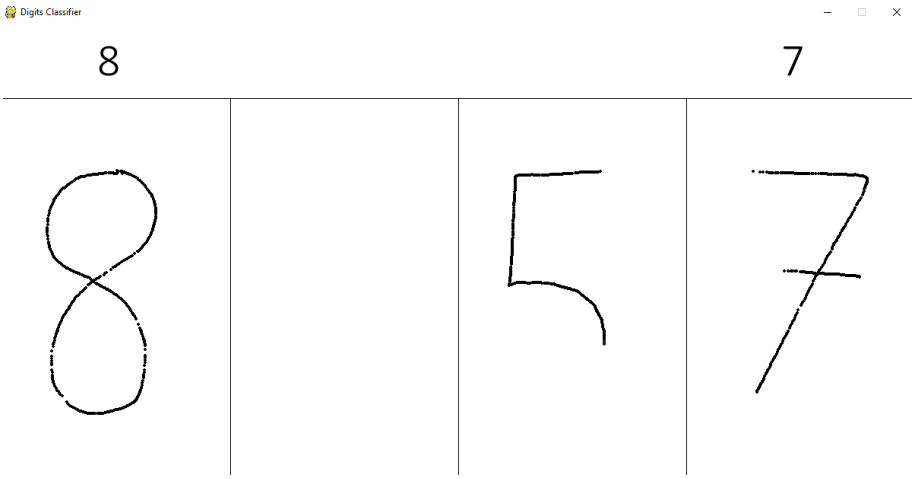


Fonte: Elaborada pelo autor

Além disso, aprimorando esse código, desenvolveu-se um algoritmo que permite a inserção e reconhecimento em tempo real de até 4 dígitos, por meio de uma tela de 1200x600 pixels. Assim, à medida em que os dígitos são escritos, a rede realiza a identificação dos mesmos e apresenta no centro da parte superior referente àquele dígito, independente da ordem em que eles são escritos.

Da mesma forma que no código anterior, ao pressionar a tecla barra de espaço, a tela é reinicializada, permitindo a inserção de novos dígitos. Na Figura 4.2, apresenta-se uma situação em que foram desenhados e reconhecidos os dígitos na 1ª e na 4ª posição e está sendo desenhado o dígito da 3ª.

Figura 4.2 – Exemplo para inserção de até 4 dígitos simultaneamente

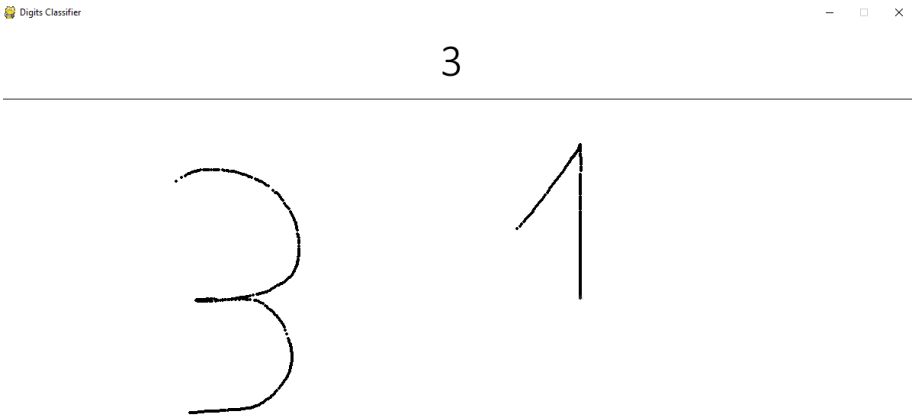


Fonte: Elaborada pelo autor

Por fim, para ilustrar melhor o funcionamento da rede, desenvolveu-se um algoritmo que gera uma tela do mesmo tamanho da anterior, 1200x600, porém sem divisões. Assim,

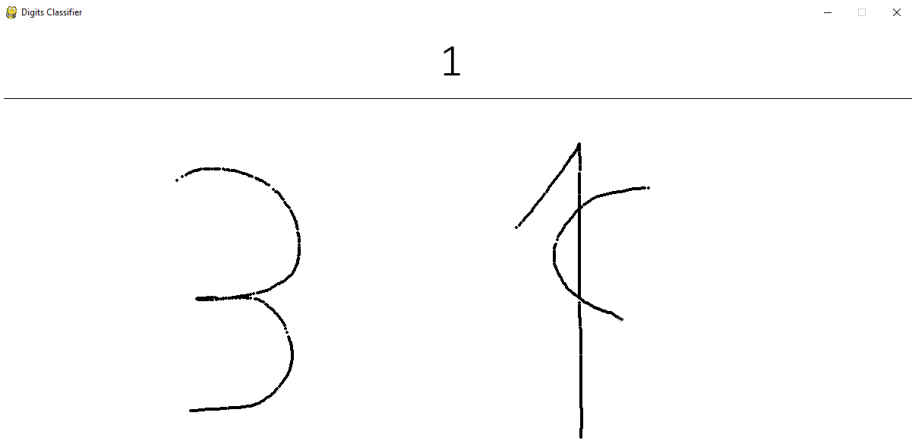
à medida em que os dígitos vão sendo escritos, a rede os reconhece e apresenta no centro superior da tela o último dígito que foi reconhecido. Nas Figuras 4.3, 4.4 e 4.5, pode-se observar o resultado da mesma para o reconhecimento dos dígitos 3, 1 e 8 que foram escritos nessa ordem.

Figura 4.3 – Reconhecimento do primeiro dígito à medida em que o segundo é escrito



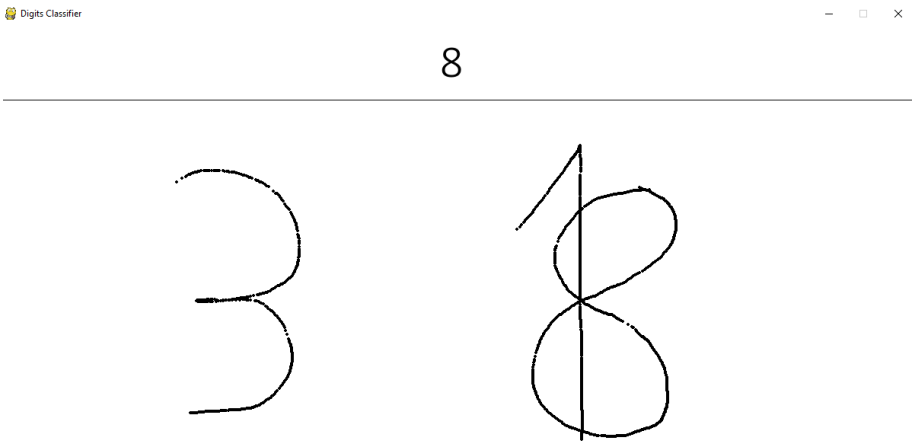
Fonte: Elaborada pelo autor

Figura 4.4 – Reconhecimento de um dígito à medida em que outro é escrito sobre ele



Fonte: Elaborada pelo autor

Figura 4.5 – Reconhecimento do último dígito correto mesmo sendo escrito junto a outro



Fonte: Elaborada pelo autor

Cabe salientar ainda, como fica claro na Figura 4.5, que a rede reconhece o dígito mesmo que ele seja escrito sobre um outro já identificado, pois são consideradas as variações das posições ao longo da escrita e não a imagem final gerada.

4.2 PRODUÇÃO DE FRAGMENTOS DE TEXTO

Os resultados obtidos com a aplicação de redes neurais na produção de fragmentos de texto serão apresentados nesta seção, abordando principalmente as informações sobre o treinamento da rede e a utilização da mesma na produção de novos fragmentos de texto.

Como já mencionado na Seção 3.2, o código para o treinamento da rede pode ser encontrado no Apêndice D. A rede foi treinada por 160 épocas, durando um total de 53 horas e cerca de 20 minutos por época. Por ser uma quantidade muito grande de épocas, são apresentadas as informações sobre precisão e erro nos dados de treinamento de algumas épocas na Tabela 4.3.

Tabela 4.3 – Dados do treinamento da segunda rede.

Época	Erro	Precisão (%)
1	2.9879	17.98
2	2.9866	17.98
3	2.9654	17.98
4	2.7295	23.29
5	2.3350	30.62
6	2.1047	36.54
7	2.0146	38.79
8	1.8889	42.49
9	1.6730	48.77
10	1.5923	51.12
15	1.3115	59.21
20	1.2093	61.54
25	1.1202	64.23
30	1.0597	66.15
35	1.0175	67.16
40	0.9816	68.28
45	0.9498	69.25
50	0.9285	69.97
55	0.9047	70.41
60	0.8803	71.20
65	0.8695	71.62
70	0.9506	69.86
75	0.9119	70.72
80	0.9733	68.84
85	0.9079	70.50
90	0.8909	71.02
95	0.8746	71.39
100	0.8000	73.71
105	0.7948	73.76
110	0.7959	73.87
115	0.8017	73.42
120	0.7921	73.79
125	0.7864	73.84
130	0.7832	74.01
135	0.7612	74.88
140	0.7607	74.55
145	0.7572	74.84
150	0.7376	75.57
155	0.7504	75.09
160	0.7352	75.63

Fonte: Elaborada pelo autor

Por ser a rede que obteve o menor erro e a melhor precisão dentre todas as épocas de treinamento, o modelo salvo e utilizado para a produção de fragmentos de texto foi o modelo obtido após as 160 épocas de treinamento. Dessa forma, a precisão obtida por esse modelo foi de 75.63%, ou seja, a rede identificava corretamente o próximo caractere na sequência aproximadamente 3 a cada 4 vezes.

Embora os resultados não pareçam tão impressionantes quanto a precisão de 99% obtida na seção anterior, cabe salientar alguns pontos referentes a essa aplicação. Primeiramente, o número de categorias a serem classificadas subiu de 10 no reconhecimento de dígitos para 56 na previsão de caracteres de texto, o que já dificulta consideravelmente a realização de uma previsão correta.

Além disso, no caso de reconhecimento de dígitos, cada sequência deveria corresponder a apenas um dígito, uma vez que a sequência corresponde ao desenho do mesmo. No caso de produção de texto, embora o treinamento considere apenas uma saída esperada, mesmas sequências de caracteres podem admitir caracteres diferentes após eles, pois diversas palavras e frases podem ser geradas a partir de um mesmo fragmento inicial de texto.

Assim, acredita-se que um índice de acertos de mais de 75% é um resultado muito interessante para essa aplicação. Com o modelo de rede neural já treinado nos conjuntos de dados, foi possível então seguir para a produção de novos fragmentos de texto.

Inicialmente, para verificar o progresso da rede ao longo do treinamento, utilizaram-se modelos correspondentes a 10, 80 e 160 épocas de treinamento. Definiu-se que o próximo caractere da sequência é o caractere que apresenta a maior probabilidade de acordo com a rede neural, uma vez que a saída da rede neural proposta é um vetor de probabilidades gerado com o auxílio da função *softmax*.

Dessa forma, definindo também que os textos gerados teriam um comprimento total de 200 caracteres, os resultados obtidos a partir desses 3 modelos tendo como base o fragmento "ainda que eu falasse as línguas dos home" podem ser observados na Tabela 4.4.

Tabela 4.4 – Fragmentos de texto produzidos pela rede.

Época	Texto gerado
10	ainda que eu falasse as línguas dos homens de todos os seus palavras de todo a todo a todo a todo a todo a todo a todo a todo a todo a todo a todo a todo a todo a todo a todo a todo a todo a todo a to
80	ainda que eu falasse as línguas dos homens de judá, e a sua fima de deus que estavará no campo de jabessericem e cinqüenta e a sua pres de simãa, e ele sej de termos de bronze, e a sua filha do se har
160	ainda que eu falasse as línguas dos homens? os teus palácios. aos seus homens de guerras de um anima, e a sua mão se regozijo. então disse o senhor a moisés: fala a teram das casas do senhor.

Fonte: Elaborada pelo autor

Ressalta-se nesse momento que a produção desses fragmentos de texto é realizada caractere a caractere, então o fato de todos os modelos conseguirem construir corretamente a palavra "homens" já comprova um bom nível de previsão mesmo com poucas épocas de treinamento.

Embora o modelo gerado após apenas 10 épocas tenha produzido apenas palavras realmente existentes, observa-se que os seus resultados não foram muito satisfatórios, uma vez que o modelo gerou repetidamente a expressão "a todo".

Já o modelo gerado após 80 épocas obteve resultados melhores, gerando realmente trechos originais, porém com diversas palavras sem sentido, como "fima", "estavará" e "jabessericem". Além disso, pode-se observar que o modelo já consegue compreender melhor o processo de escrita, compreendendo que deve haver um espaçamento entre as vírgulas e a palavra seguinte.

Por fim, o modelo gerado após 160 épocas surpreende na medida em que apresenta uma estrutura muito semelhante a nossa escrita, colocando espaçamentos ou novas linhas após sinais de pontuação como pontos de interrogação, pontos finais, dois pontos e vírgulas. Aliado a isso, embora o texto em si não possua um sentido completo, apenas duas palavras produzidas pelo modelo não são encontradas nos dados de treinamento ("anima" e "teram"), mostrando que a rede realmente conseguiu produzir textos interessantes caractere a caractere.

Após isso, utilizou-se apenas o modelo gerado após 160 épocas, mas a produção de fragmentos de texto foi realizada de duas maneiras distintas. Sabe-se que a saída da rede neural é um vetor de probabilidades contendo as probabilidades de o próximo dígito

ser cada um dos 56 caracteres presentes no texto original.

Dessa forma, o primeiro método de treinamento corresponde ao método já apresentado de simplesmente assumir que o caractere que apresenta a maior probabilidade de acordo com a rede neural. Enquanto isso, o segundo método consiste em considerar que o próximo dígito da sequência pode corresponder a qualquer um dos 56 caracteres de acordo com a probabilidade apresentada no vetor de saída.

Assim, partindo do modelo de rede neural já treinado e com um fragmento inicial de 40 caracteres pré-determinado, a primeira metodologia apresentará sempre os mesmos resultados de forma determinística, enquanto a segunda metodologia poderá fornecer resultados diferentes em cada aplicação de forma probabilística.

Com isso, para diversos fragmentos retirados do banco de dados de treinamento foi gerado um fragmento de texto de 200 caracteres utilizando a primeira metodologia juntamente com dois textos gerados pela segunda metodologia. A partir disso, foram selecionados dois desses fragmentos e os textos gerados a partir deles são apresentados na Tabela 4.5, onde o primeiro fragmento gerado refere-se ao primeiro método e os outros dois referem-se ao segundo método.

Tabela 4.5 – Fragmentos de texto gerados pela rede.

Trecho inicial	Texto gerado
agora, pois, permanecem a fé, a esperanç	agora, pois, permanecem a fé, a esperança de deus se levantará como o rei da assíria, como também o meu santuário nem para suprisse todos os primogênitos: assim diz o senhor dos exércitos: a quem me d
	agora, pois, permanecem a fé, a esperança na terra, a terrificação do tabernáculo que já não seja pinte se eternitas que se pôs a ti ungeriavia na mão dos seus corações. o seu sangue e diante do rib
	agora, pois, permanecem a fé, a esperança andares, no dia em que tinha fortaleces as ti, e será pereceu, por toda a terra, e será conforme a fala do senhor. subiu e a justiça. então decretando dos
porta da tenda, e todo o povo, levantand	porta da tenda, e todo o povo, levantando o seu coração os teus servos, e a sua oferta de cereais e o seu povo israel.
	e a sua mão em pé disseram: deus e a sua voz alguns de servos de seus pais, e a
	porta da tenda, e todo o povo, levantando-se da presença dos aclasias. portanto se assim diz o senhor: seu filho, e porção de fazer-lhe jesus, filho do homem, para que jacontou-lhe pedra e a tirreme
	porta da tenda, e todo o povo, levantando o rei, anda os magas para as gerações. acaso comer o corpo da tua mão; na fé destruída são desta casa de caminar a provassa todo o exército dois vela. s

Fonte: Elaborada pelo autor

A partir dos fragmentos de textos apresentados, observa-se que os textos gerados pelo primeiro método são bem mais precisos, gerando palavras conhecidas, mas contando

com a desvantagem de que, dado um fragmento de texto inicial, o texto final gerado será sempre o mesmo.

Por outro lado, a segunda metodologia possibilita uma variedade maior de textos gerados, porém estes apresentam uma quantidade maior de palavras desconhecidas, visto que cada caractere inserido tem probabilidade de assumir diversos caracteres, não optando sempre pelo caractere mais adequado.

Por fim, para mostrar que a rede é capaz de construir textos maiores evitando repetições como as apresentadas na Tabela 4.4 para o modelo gerado após 10 épocas, é apresentado a seguir um fragmento de texto de 1000 caracteres gerado pela rede neural, tendo como base o trecho "filho do homem, eis que porão cordas sob".

"filho do homem, eis que porão cordas sobre a terra.

então disse jacó de todo o povo que se deste ao homem nos céus, e a terra que o senhor teu deus te alegre, dizendo: o rei ao saul contra eles a tua benignidade, a figuestão do senhor, dizendo:

falai-o a terra do egito, até honra de seu pai, e o senhor teu deus te alegre, e a sua mão se levanta da casa do senhor, e a tua benignidade da casa do senhor, e a sua face da terra do egito, até o dia do teu povo israel.

e a tua mão destruireis de babilônia a teu redor provi-lhe eles recusarei as cidades da minha palavra de deus;

omem que se ajuntarão a teu redor de todos os seus cavalos.

ora, havemos dos anciãos de israel, poder de farinha aos meus olhos, e as suas tendas comensias testemunhos o seu pecado, e a sua oferta de cereais, e os filhos de israel não tem nós o que tal entre vós estas coisas aos outros homens de gordura da casa do senhor, dizendo:

falai-vos a sua salva, e ele a sua famílias ele a tua maldade, e a"

Dessa forma, pode-se concluir mais uma vez que a rede neural conseguiu reconstruir diversas das palavras que estavam nos dados de treinamento e gerou fragmentos de texto originais, não havendo repetições de longos trechos do conjunto de dados. Novamente, salienta-se também que a rede foi capaz de produzir fragmentos de textos com frases não muito longas e respeitando os espaçamentos após pontuações, apresentando a eficiência da rede neural na compreensão e reprodução de alguns elementos essenciais da escrita.

4.3 PREVISÕES DE NÍVEIS DE POLUIÇÃO

Nesta seção serão abordados os resultados alcançados com a aplicação de redes neurais artificiais na previsão de índices de poluição, tendo como base o banco de dados Beijing PM2.5. O código utilizado para o treinamento da rede pode ser observado no Apêndice F.

O treinamento durou 12 horas, produzindo modelos de 103 épocas distintas e com um tempo médio de treinamento de 7 minutos por época. Devido ao grande número de épocas, foram apresentadas as informações sobre erro absoluto médio e erro quadrático médio de algumas épocas na Tabela 4.6.

Tabela 4.6 – Dados do treinamento da terceira rede.

Época	Erro absoluto médio	Erro quadrático médio
1	0.0315	0.002946
2	0.0258	0.002005
3	0.0223	0.001662
4	0.0188	0.001316
5	0.0173	0.001151
6	0.0168	0.001104
7	0.0166	0.001074
8	0.0153	0.000969
9	0.0145	0.000952
10	0.0142	0.000898
15	0.0125	0.000819
20	0.0111	0.000755
25	0.0103	0.000733
30	0.0099	0.000720
35	0.0101	0.000723
40	0.0096	0.000712
45	0.0095	0.000705
50	0.0093	0.000700
55	0.0094	0.000707
60	0.0093	0.000699
65	0.0092	0.000698
70	0.0094	0.000705
75	0.0092	0.000691
80	0.0092	0.000699
85	0.0093	0.000691
90	0.0094	0.000702
95	0.0090	0.000692
100	0.0093	0.000693
103	0.0090	0.000683

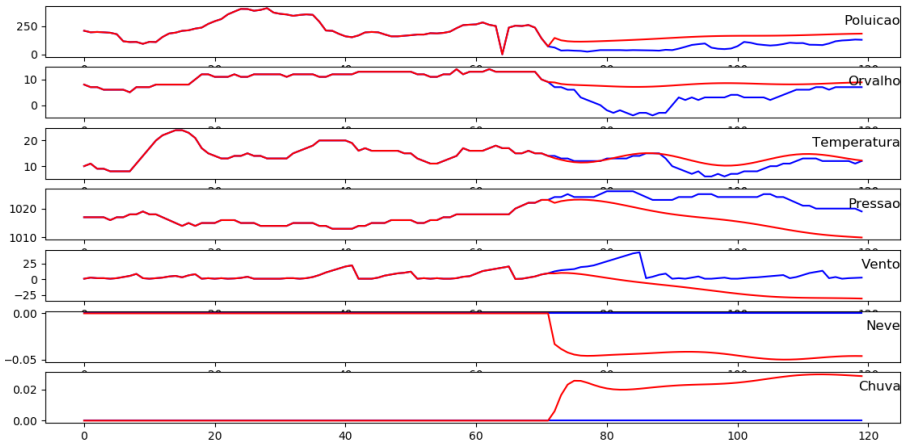
Fonte: Elaborada pelo autor

O modelo selecionado para a previsão de níveis de poluição foi o modelo gerado após 103 épocas de treinamento, pois foi o modelo que obteve o menor erro absoluto médio e menor erro quadrático médio. Embora essas medidas não apresentem uma noção tão clara da eficiência da rede como a precisão, pode-se observar a grande redução desses erros ao longo do treinamento da rede.

Para ilustrar melhor essa evolução, nas Figuras 4.6 e 4.7 são apresentadas as previsões da rede após 10 épocas de treinamento e após 103 épocas de treinamento para um mesmo conjunto de dados iniciais. Em cada um dos testes, foi utilizada uma sequência de 72 horas de dados e a rede buscou prever as informações de poluição, orvalho, tem-

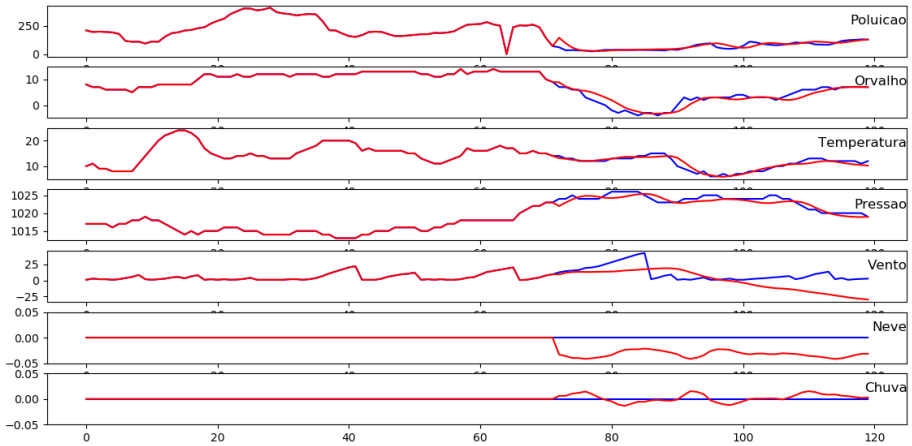
peratura, pressão, vento, neve e chuva das 48 horas seguintes, correspondendo a 3 e 2 dias, respectivamente. Em azul, são apresentados os dados originais e na cor vermelha a previsão gerada pela rede.

Figura 4.6 – Resultado após 10 épocas



Fonte: Elaborada pelo autor

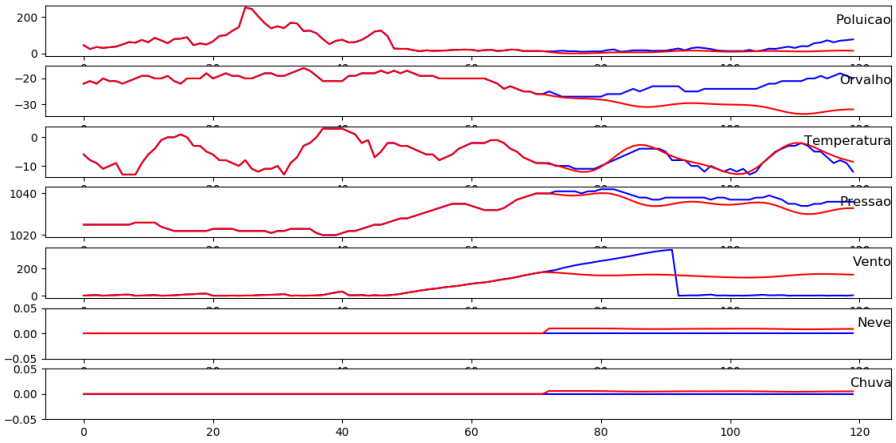
Figura 4.7 – Resultado após 103 épocas



Fonte: Elaborada pelo autor

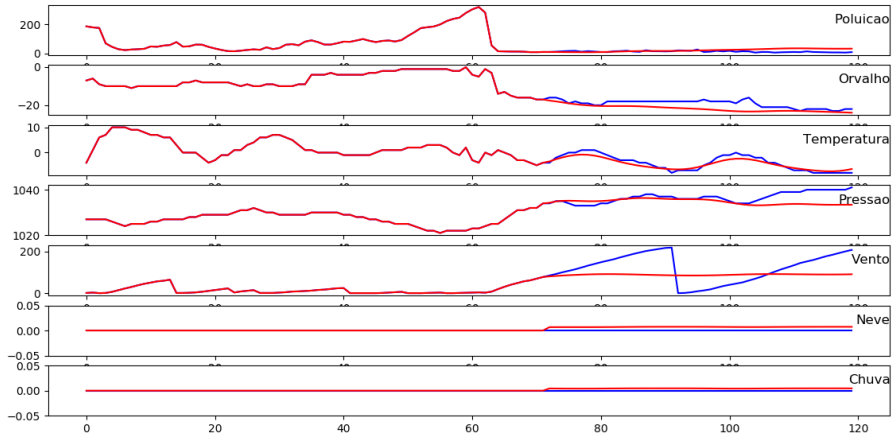
Assim, fica claro como a rede realmente obteve resultados mais satisfatórios à medida que o treinamento foi avançando. Com isso, utilizando o modelo de rede neural treinado durante 103 épocas, os resultados de algumas previsões são apresentados nas Figuras 4.8 e 4.9.

Figura 4.8 – Exemplo de resultados de previsão



Fonte: Elaborada pelo autor

Figura 4.9 – Exemplo de resultados de previsão

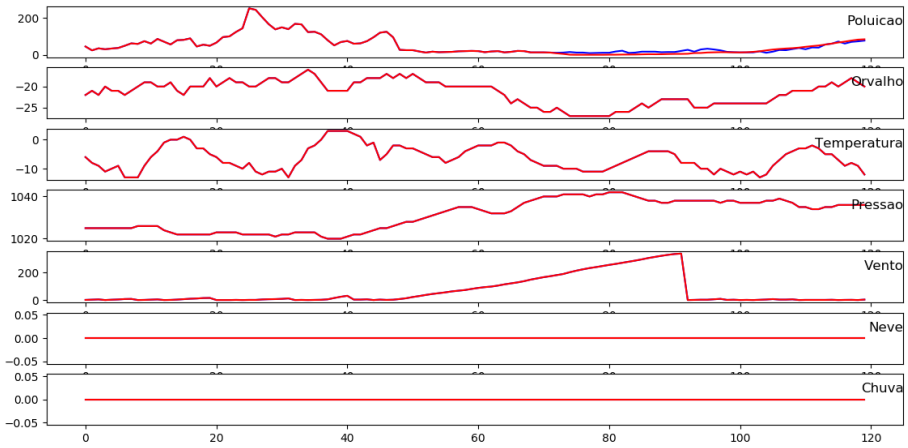


Fonte: Elaborada pelo autor

Observa-se que, utilizando apenas as informações das 72 horas iniciais, a rede apresenta alguma dificuldade para prever as mudanças bruscas de variação no vento, por exemplo. Ainda assim, de forma geral, os resultados se assemelham bastante com os valores esperados.

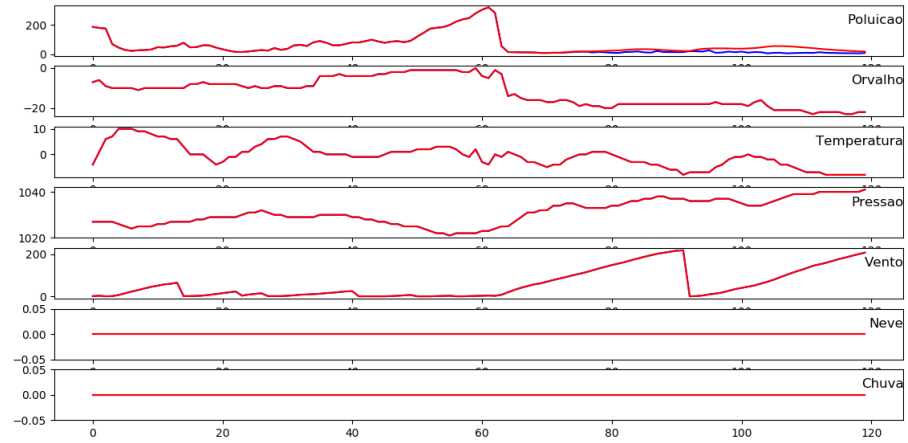
Em seguida, com o intuito de fazer com que a rede produzisse resultados mais semelhantes com a realidade, além de utilizar as informações das 72 horas iniciais, a rede também foi alimentada com todas as informações de orvalho, temperatura, pressão, vento, neve e chuva. Dessa forma, assim que a rede neural realizava uma previsão de todas as informações, os valores dessas 7 características eram atualizadas para os valores corretos, fazendo com que a previsão realmente fosse apenas dos índices de poluição. Os novos resultados gerados com o mesmo conjunto de dados iniciais das Figuras 4.8 e 4.9 são agora apresentados nas Figuras 4.10 e 4.11.

Figura 4.10 – Exemplo de previsão de poluição



Fonte: Elaborada pelo autor

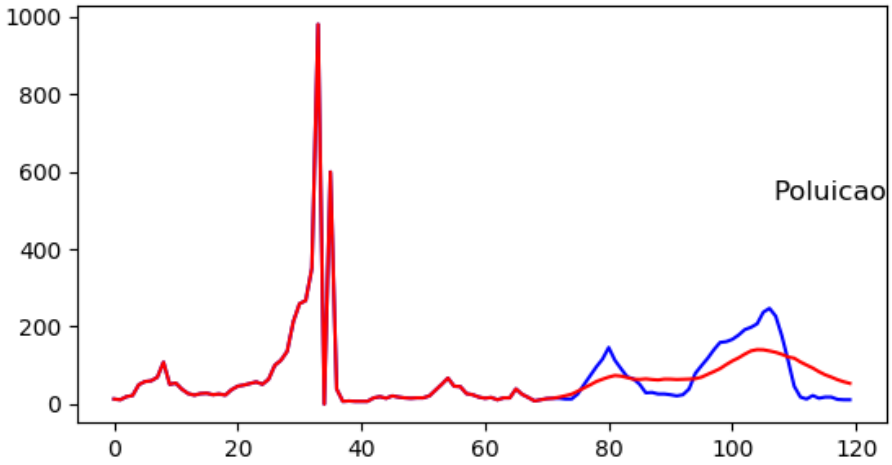
Figura 4.11 – Exemplo de previsão de poluição



Fonte: Elaborada pelo autor

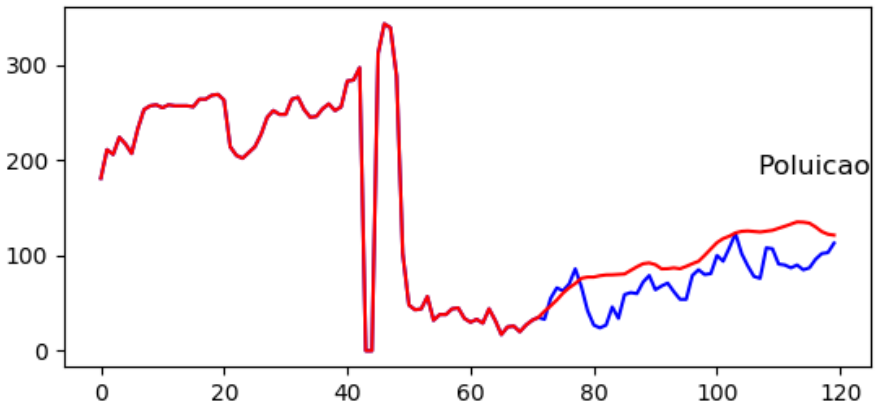
Como nessas situações, todas as informações se repetem com os dados corretos, com exceção da poluição, as Figuras 4.12 e 4.13 foram construídas para apresentar mais alguns resultados de previsão de níveis de poluição gerados pela rede, desta vez apresentando apenas o primeiro gráfico.

Figura 4.12 – Previsão de índices de poluição



Fonte: Elaborada pelo autor

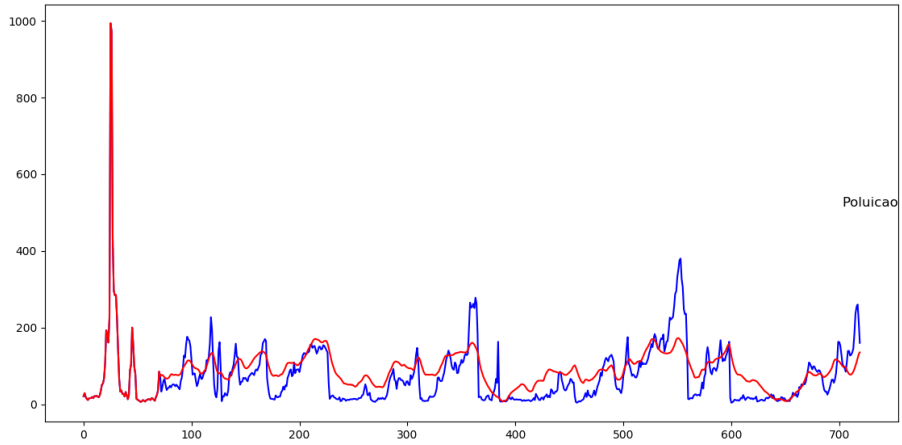
Figura 4.13 – Previsão de índices de poluição



Fonte: Elaborada pelo autor

Esses exemplos comprovam que a rede consegue prever satisfatoriamente os índices de poluição a curto prazo, porém seria interessante que a rede obtivesse bons resultados também a longo prazo. Dessa forma, na Figura 4.14 são apresentados os resultados de uma previsão de 30 dias, tempo esse que inclui os 3 dias de dados iniciais fornecidos para a rede.

Figura 4.14 – Previsão de índices de poluição



Fonte: Elaborada pelo autor

Por fim, observa-se também que a rede neural apresentou uma boa capacidade de previsão do índice proposto até mesmo a longo prazo, conseguindo reproduzir diversos picos de poluição e com resultados bem próximos dos dados presentes no bancos de dados utilizado.

4.4 REALIZAÇÃO DE MOVIMENTOS NO JOGO PONG

Na seção final desde capítulo será abordado o jogo Pong e todos os resultados alcançados por meio da utilização de redes neurais artificiais na previsão de seus movimentos. No Apêndice I é apresentado o código utilizado nesta aplicação.

A rede neural utilizada para essa aplicação foi treinada por 133 épocas, durando um total de 13 horas e uma média de 6 minutos por época. Novamente, são apresentadas algumas informações sobre erro absoluto médio e erro quadrático na Tabela 4.7.

Tabela 4.7 – Dados do treinamento da quarta rede.

Época	Erro absoluto médio	Erro quadrático médio
1	0.0376	0.0139
2	0.0327	0.0134
3	0.0314	0.0132
4	0.0323	0.0130
5	0.0309	0.0126
6	0.0300	0.0125
7	0.0296	0.0121
8	0.0294	0.0115
9	0.0295	0.0113
10	0.0278	0.0111
15	0.0285	0.0103
20	0.0257	0.0095
25	0.0230	0.0087
30	0.0226	0.0080
35	0.0224	0.0075
40	0.0217	0.0074
45	0.0211	0.0071
50	0.0198	0.0068
55	0.0193	0.0065
60	0.0187	0.0062
65	0.0187	0.0061
70	0.0177	0.0059
75	0.0192	0.0060
80	0.0176	0.0059
85	0.0181	0.0059
90	0.0185	0.0060
95	0.0181	0.0058
100	0.0182	0.0058
105	0.0184	0.0057
110	0.0196	0.0058
115	0.0197	0.0057
120	0.0182	0.0056
125	0.0189	0.0056
130	0.0172	0.0054
133	0.0173	0.0053

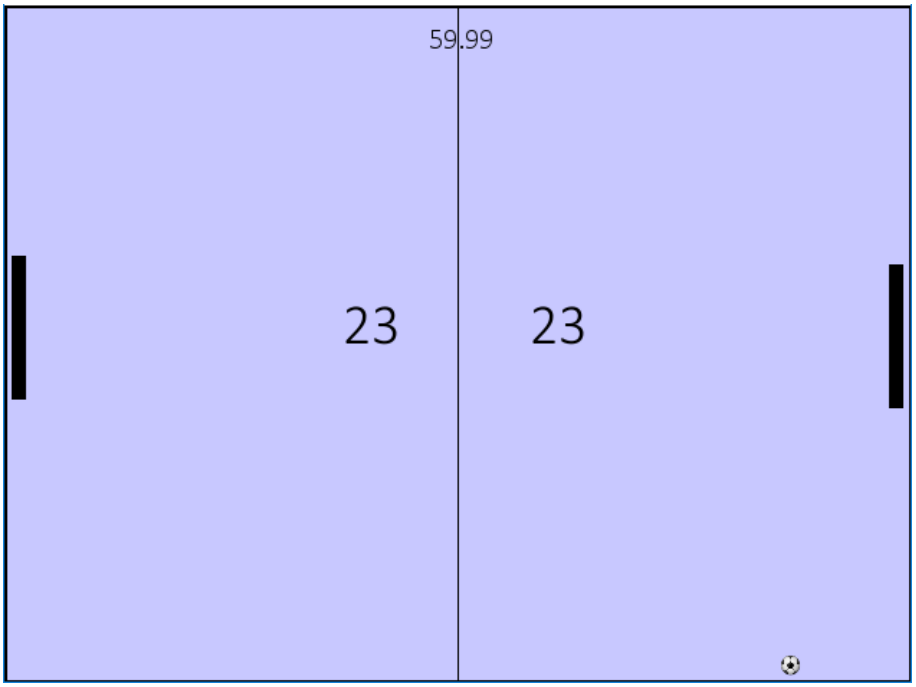
Fonte: Elaborada pelo autor

Nesse caso, os modelos gerados após 130 e após 133 épocas de treinamento foram os que obtiveram melhores resultados, apresentando os menores erros absoluto médio e quadrático médio, respectivamente. Dessa forma, os resultados serão apresentados para esses dois modelos distintos que serão denominados simplesmente Modelos 1 e 2,

respectivamente.

O método de avaliação utilizado para esses modelos foi o resultado de 60s de jogo. Assim, tomou-se como base de comparação o resultado obtido em 60s de jogo com ambas as barras se movimentando de maneira aleatória. O resultado final desse jogo foi 23 a 23, como pode ser observado na Figura 4.15.

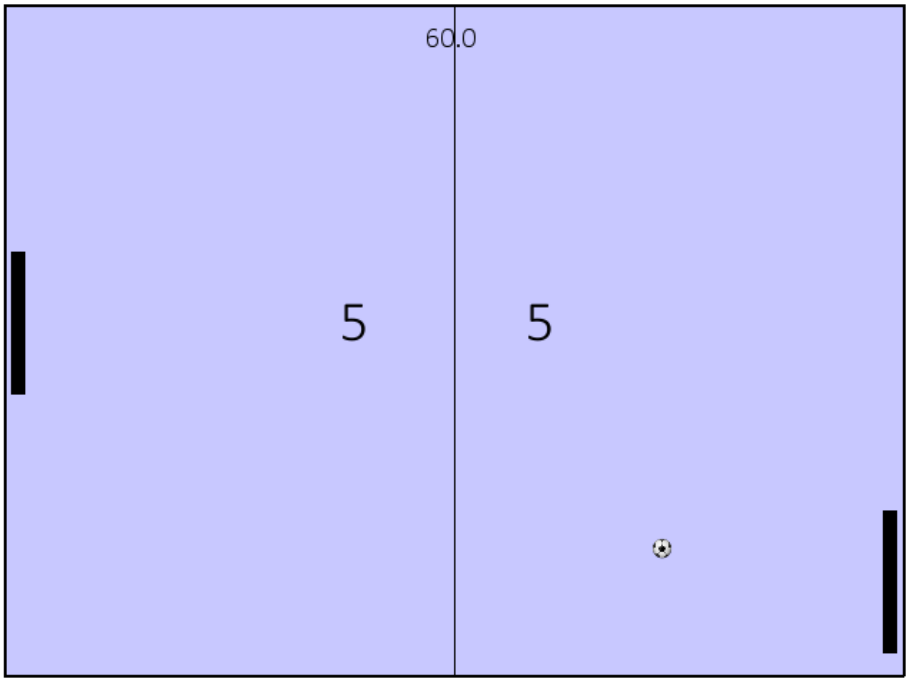
Figura 4.15 – Resultado de um jogo aleatório de Pong



Fonte: Elaborada pelo autor

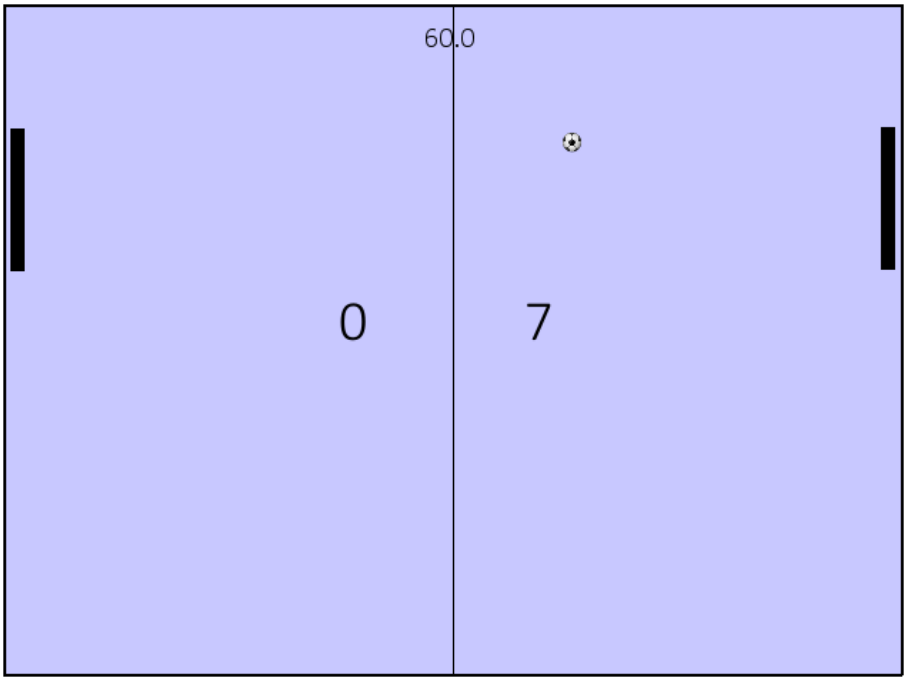
Tendo esse resultado como base, ambos os modelos foram testados para verificar o resultado após 60s. Tendo seus movimentos baseados no Modelo 1, o resultado final foi de 5 a 5, enquanto com o Modelo 2 o resultado final foi de 0 a 7, como pode ser observado nas Figuras 4.16 e 4.17.

Figura 4.16 – Resultado do Modelo 1 no Pong



Fonte: Elaborada pelo autor

Figura 4.17 – Resultado do Modelo 2 no Pong



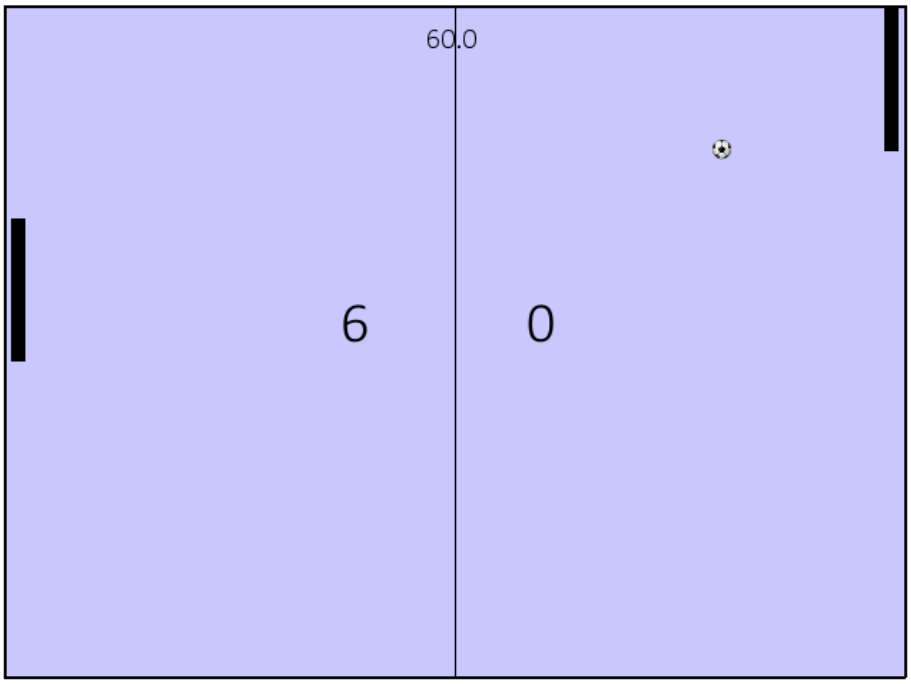
Fonte: Elaborada pelo autor

Embora os resultados não pareçam muito promissores, quando comparados com o teste para movimentos aleatórios, o número de pontos sofridos foi reduzido entre 75 e

85%, diminuindo de 46 para 10 no Modelo 1 e para 6 no Modelo 2. Dessa forma, verifica-se que de fato o treinamento da rede contribuiu para o aprendizado do funcionamento do jogo.

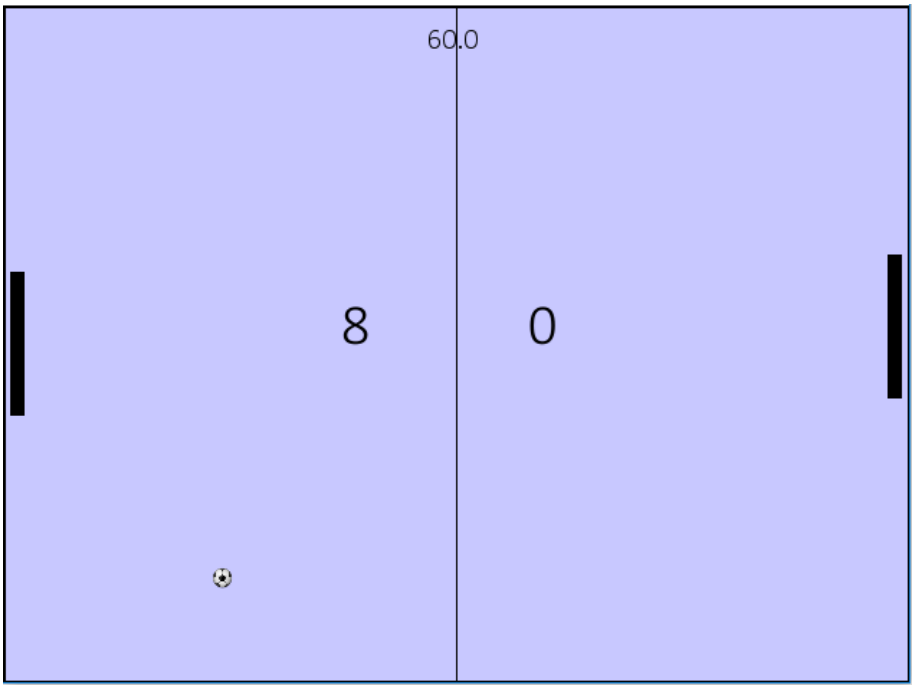
Além disso, outro teste que foi realizado com ambos os modelos foi a realização de uma partida de 60s contra um adversário controlado pelo autor desse trabalho. Assim, quando a rede neural gerava o movimento apenas da segunda barra, os resultados foram de 6 a 0 para o Modelo 1 e de 8 a 0 para o Modelo 2, conforme é apresentado nas Figuras 4.18 e 4.19.

Figura 4.18 – Resultado do Modelo 1 no Pong



Fonte: Elaborada pelo autor

Figura 4.19 – Resultado do Modelo 2 no Pong



Fonte: Elaborada pelo autor

Embora resultados melhores pudessem ter sido obtidos com algoritmos mais simples, é interessante o fato de que a rede consiga compreender de certa forma o funcionamento do jogo Pong simplesmente a partir da análise de movimentos de outro jogador. Com isso, observa-se que a rede foi capaz de obter bons resultados a partir do treinamento do Pong, comprovando mais uma vez a eficiência da utilização de redes neurais artificiais nas mais diversas aplicações.

5 CONCLUSÕES

As redes neurais implementadas foram capazes de realizar todas as atividades propostas de maneira satisfatória. Os modelos implementados conseguiram realizar o reconhecimento de dígitos em tempo real, produzir fragmentos de texto, prever índices de poluição e realizar movimentos no jogo Pong, de acordo com o que foi proposto.

Além disso, as redes LSTM apresentaram razoável robustez nos diversos testes que foram realizados em cada uma das aplicações desse trabalho. Destaca-se aqui o reconhecimento de dígitos independentemente da posição em que ele era escrito ou de estar sendo escrito sobre outro dígito ou sobre uma tela branca, a produção de muitas palavras existentes em grandes fragmentos de texto, a boa previsão de até 30 dias de informações referentes à poluição e os resultados consideráveis na realização de 60s de movimentos no jogo Pong.

Além disso, cabe ressaltar que os métodos apresentados ao longo deste trabalho e os algoritmos em anexo podem ser facilmente adaptados para serem utilizados em diversas outras aplicações, bastando que se obtenha um banco de dados adequado. Esse fato, como já pode ser observado no desenvolvimento de todo o trabalho, contribui para verificar a versatilidade na utilização de redes neurais.

Isso posto, conclui-se que de fato as redes neurais artificiais LSTM garantem uma ótima versatilidade e, portanto, acredita-se que elas são uma ferramenta que ainda pode ser muito explorada em diversas outras aplicações.

REFERÊNCIAS BIBLIOGRÁFICAS

BROWNLEE, J. **Text Generation With LSTM Recurrent Neural Networks in Python with Keras**. August 2016. Acesso em: jan. 2018. Disponível em: <https://machinelearningmastery.com/text-generation-lstm-recurrent-neural-networks-python-keras/>.

CHELLAPILLA, K.; FOGEL, D. B. Evolution, neural networks, games, and intelligence. **Proceedings of the IEEE**, v. 87, n. 9, p. 1471–1496, 1999.

CHOLLET, F. et al. **Keras**. [S.l.]: GitHub, 2015. https://keras.io.

GREFF, K. et al. LSTM: A search space odyssey. **IEEE transactions on neural networks and learning systems**, v. 28, n. 10, p. 2222–2232, 2017.

HINTON, G. E.; OSINDERO, S.; TEH, Y.-W. A fast learning algorithm for deep belief nets. **Neural computation**, v. 18, n. 7, p. 1527–1554, 2006.

KOK, I.; SIMSEK, M. U.; ÖZDEMIR, S. A deep learning model for air quality prediction in smart cities. In: **2017 IEEE International Conference on Big Data (Big Data)**. [S.l.: s.n.], 2017. p. 1983–1990.

KOVACS, Z. L. **Redes neurais artificiais**. [S.l.]: Editora Livraria da Física, 2002.

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

OLAH, C. **Understanding LSTM Networks**. [S.l.]: GitHub, 2015. http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

RAUBER, T. W. **Redes neurais artificiais**. [S.l.]: Universidade Federal do Espírito Santo, 2005.

RAVAL, S. **How to Generate Music - Intro to Deep Learning 9**. March 2017. Acesso em: jan. 2018. Disponível em: <https://www.youtube.com/watch?v=4DMm5Lhey1U>.

SAK, H.; SENIOR, A.; BEAUFAYS, F. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. **Fifteenth Annual Conference of the International Speech Communication Association**, 2014.

SCHMIDHUBER, J. Our impact on the world's most valuable public companies. April 2017. Acesso em: jun. 2017. Disponível em: <http://people.idsia.ch/ juergen/impact-on-most-valuable-companies.html>.

SIMARD, P. Y. et al. Best practices for convolutional neural networks applied to visual document analysis. **ICDAR**, p. 958–962, 2003.

SINHA, A. **An improved recognition module for the identification of handwritten digits**. 2002. Tese (Doutorado), 2002.

SUNDERMEYER, M.; NEY, H.; SCHLüTER, R. From feedforward to recurrent lstm neural networks for language modeling. **IEEE Transactions on Audio, Speech, and Language Processing**, v. 23, n. 3, p. 517–529, 2015.

TOLOSANA, R. et al. Exploring recurrent neural networks for on-line handwritten signature biometrics. **IEEE Access**, v. 6, p. 5128–5138, 2018.

TRASK, I. **Anyone can learn to code an LSTM-RNN in Python (Part 1: RNN)**. November 2015. Acesso em: jun. 2017. Disponível em: <http://iamtrask.github.io/2015/11/15/anyone-can-code-lstm/>.

VEEN, F. V. **The Neural Network Zoo**. September 2016. Acesso em: jun. 2017. Disponível em: <http://www.asimovinstitute.org/neural-network-zoo/>.

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

APÊNDICE A – CÓDIGO DE PROCESSAMENTO DOS DADOS PARA RECONHECIMENTO DE DÍGITOS

```
import pickle

''' Carrega os dados com posições e rotulos dos dados de treinamento,
validacao e teste '''
X_train, labels_train = pickle.load(open( "pos_train.p", "rb" ))
X_valid, labels_valid = pickle.load(open( "pos_valid.p", "rb" ))
X_test, labels_test = pickle.load(open( "pos_test.p", "rb" ))

''' Inicializa como vazios os vetores que serao utilizados para medir o
comprimento maximo e para gerar o banco de dados '''
comp=[]
x_train = [[]]
x_valid = [[]]
x_test = [[]]

''' Define cada elemento dos dados finais de treinamento como a diferenca
de duas posicoes consecutivas dos dados apenas com as posicoes, gravando
tambem os comprimentos de cada vetor '''
for i in range(len(X_train)):
    for j in range(1,len(X_train[i])):
        elemento =  [X_train[i][j][0]-X_train[i][j-1][0],X_train[i][j][1]-
        X_train[i][j-1][1]]
        x_train[i].append(elemento)
    comp.append(len(x_train[i]))
    x_train.append([])
del(x_train[-1])

''' Realiza o mesmo procedimento para os dados de validacao e teste '''
for i in range(len(X_valid)):
    for j in range(1,len(X_valid[i])):
        elemento =  [X_valid[i][j][0]-X_valid[i][j-1][0],X_valid[i][j][1]-
        X_valid[i][j-1][1]]
        x_valid[i].append(elemento)
    comp.append(len(x_valid[i]))
    x_valid.append([])
```

```
del(x_valid[-1])

for i in range(len(X_test)):
    for j in range(1,len(X_test[i])):
elemento = [X_test[i][j][0]-X_test[i][j-1][0],X_test[i][j][1]-
        X_test[i][j-1][1]]
        x_test[i].append(elemento)
    comp.append(len(x_test[i]))
    if(len(x_test[i]))>600:
        print(i)
    x_test.append([])
del(x_test[-1])

''' Determina o comprimento maximo dentre todos os vetores '''
comprimento = max(comp)
print 'O comprimento final dos vetores eh de', comprimento

''' Completa cada um dos vetores com variações nulas até o comprimento
máximo calculado e inverte os mesmos '''
for i in range(len(x_train)):
    for j in range(len(x_train[i]),comprimento):
        x_train[i].append([0, 0])
    x_train[i].reverse()

for i in range(len(x_valid)):
    for j in range(len(x_valid[i]),comprimento):
        x_valid[i].append([0, 0])
    x_valid[i].reverse()

for i in range(len(x_test)):
    for j in range(len(x_test[i]),comprimento):
    x_test[i].append([0, 0])
    x_test[i].reverse()

''' Salva os arquivos para gerar o banco de dados processado '''
pickle.dump([x_train, labels_train], open( "data_train.p", "wb" ) )
pickle.dump([x_valid, labels_valid], open( "data_valid.p", "wb" ) )
pickle.dump([x_test, labels_test], open( "data_test.p", "wb" ) )
print('Todos os arquivos foram salvos!')
```

APÊNDICE B – CÓDIGO DE TREINAMENTO DA PRIMEIRA REDE

```
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau,
EarlyStopping
from keras.models import Sequential, load_model
from keras.layers import LSTM, Dense
from keras.utils import to_categorical
import numpy as np
import pickle

''' Carrega os dados de treinamento e validacao '''
x_train, labels = pickle.load(open( "data_train.p", "rb" ))
x_valid, labels_valid = pickle.load(open( "data_valid.p", "rb" ))

''' Transforma os rotulos para o formato de one-hot encoding '''
one_hot_labels = to_categorical(labels, num_classes=10)
one_hot_labels_valid = to_categorical(labels_valid, num_classes=10)

print('Dados prontos!')

''' Define que o modelo sera uma sequencia linear de camadas '''
model = Sequential()

''' Cada comando model.add(LSTM()) adiciona uma camada da rede recorrente
LSTM ao modelo '''
model.add(LSTM(32, return_sequences=True,
               input_shape=(600, 2)))
print('Camada 1 pronta!')
model.add(LSTM(32, return_sequences=True))
print('Camada 2 pronta!')
model.add(LSTM(32))
print('Camada 3 pronta!')

''' Adiciona uma camada final que aplica a funcao softmax para gerar um
vetor de probabilidades '''
model.add(Dense(10, activation='softmax'))
print('Camadas prontas!')
```

```
''' Compila o modelo, definindo o calculo do erro, o otimizador e a
metrica '''
model.compile(loss='categorical_crossentropy', optimizer='rmsprop',
metrics=['accuracy'])
print('Modelo compilado!')

''' Grava apenas o modelo com menor erro de validacao ao fim de cada
epoca '''
checkpointer = ModelCheckpoint(filepath="modelo5.h5", verbose=0,
save_weights_only=False, save_best_only=True)
''' Reduz a taxa de aprendizagem se o erro de validacao nao diminuir em
duas epocas consecutivas '''
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=1,
min_lr=0.001)
''' Encerra o treinamento antecipadamente se o erro de validacao nao
diminuir por seis epocas consecutivas '''
earlyst = EarlyStopping(monitor='val_loss', min_delta=0, patience=5,
verbose=0, mode='auto')

''' Realiza o treinamento da rede, definindo os dados de treinamento e
validacao, assim como o numero maximo de epocas e as funcoes auxiliares
utilizadas '''
model.fit(x_train, one_hot_labels,
          batch_size=32, epochs=100, verbose=1,
          validation_data=(x_valid, one_hot_labels_valid),
          callbacks=[checkpointer, reducelr, earlyst])
print('Modelo treinado!')

''' Salva o modelo apos treinamento '''
model.save('my_model5.h5')
print('Modelo salvo!')
```

APÊNDICE C – CÓDIGO PARA TESTE DO PRIMEIRO MODELO

```
from keras.models import Sequential, load_model
from keras.layers import LSTM, Dense
from keras.utils import to_categorical
import numpy as np
import pickle

''' Carrega os dados de treinamento, validacao e teste '''
x_test, labels_test = pickle.load(open( "data_test.p", "rb" ))

''' Transforma os rotulos para o formato de one-hot encoding '''
one_hot_labels_test = to_categorical(labels_test, num_classes=10)
print('Dados prontos!')

''' Carrega o modelo de rede neural treinado anteriormente '''
model=load_model('my_model.h5')
print('Teste para o modelo: ')

''' Preve os resultados para os dados de teste '''
resultados = model.predict(x_test)
''' Verifica o total de amostras a serem analisadas '''
total = len(x_test)
''' Conta a quantidade de acertos nos dados testados e apresenta
os digitos identificados incorretamente '''
corretos = 0
for i in range(len(resultados)):
    if labels_test[i] == np.argmax(resultados[i]):
        corretos += 1
    else:
        print(labels_test[i], np.argmax(resultados[i]))

''' Calcula a precisao da rede nos dados apresentados '''
precisao = 100.*corretos/total
print("A precisao nos dados de teste foi de %.2f %%" %precisao)
```

APÊNDICE D – CÓDIGO DE TREINAMENTO DA SEGUNDA REDE

```
from __future__ import print_function
from keras.models import Sequential, load_model
from keras.layers import Dense, Activation
from keras.layers import LSTM
from keras.optimizers import RMSprop
from keras.utils.data_utils import get_file
import numpy as np
import codecs

''' Carrega o arquivo de texto e transforma em letras minusculas '''
arquivo = codecs.open('biblia.txt', 'r', encoding='latin1')
text=arquivo.read().lower()
print('Comprimento do texto:', len(text))

''' Enumera os caracteres presentes no texto '''
chars = sorted(list(set(text)))
print('Total de caracteres:', len(chars))
char_indices = dict((c, i) for i, c in enumerate(chars))
indices_char = dict((i, c) for i, c in enumerate(chars))


''' Divide o texto em diversas sequencias de comprimento 40, tomando um
intervalo de 10 caracteres entre as sequencias '''
maxlen = 40
step = 10
sentences = []
next_chars = []
for i in range(0, len(text) - maxlen, step):
    sentences.append(text[i: i + maxlen])
    next_chars.append(text[i + maxlen])
print('Total de sequencias:', len(sentences))

''' Transforma cada uma das sequencias em matrizes no formato one-hot
encoding '''
X = np.zeros((len(sentences), maxlen, len(chars)), dtype=np.bool)
y = np.zeros((len(sentences), len(chars)), dtype=np.bool)
```

```
for i, sentence in enumerate(sentences):
    for t, char in enumerate(sentence):
        X[i, t, char_indices[char]] = 1
    y[i, char_indices[next_chars[i]]] = 1


''' Cria o modelo de rede neural com 2 camadas LSTM alternadas com 2
camadas Dropout '''
model = Sequential()
model.add(LSTM(128, return_sequences=True,input_shape=(maxlen, len(chars))))
model.add(Dropout(0.2))
model.add(LSTM(128))
model.add(Dropout(0.2))
model.add(Dense(len(chars)))
model.add(Activation('softmax'))
optimizer = RMSprop(lr=0.01)
model.compile(loss='categorical_crossentropy', metrics=['accuracy'],
optimizer=optimizer)

''' Realiza o treinamento da rede, salvando os modelos ao fim de cada
iteracao '''
for iteration in range(1,200):
    print()
    print('-' * 50)
    print('Iteracao', iteration)
    model.fit(X, y, batch_size=8192, epochs=1)
    model.save('modelo_biblia_'+str(iteration)+'.h5')
```

APÊNDICE E – CÓDIGO PARA TESTE DO SEGUNDO MODELO

```
from __future__ import print_function
from keras.models import Sequential, load_model
from keras.layers import Dense, Activation
from keras.layers import LSTM
from keras.optimizers import RMSprop
from keras.utils.data_utils import get_file
import sys
import random
import numpy as np
import codecs

''' Carrega o arquivo de texto e transforma em letras minusculas '''
arquivo = codecs.open('biblia.txt', 'r', encoding='latin1')
text=arquivo.read().lower()
print('Comprimento do texto:', len(text))

''' Enumera os caracteres presentes no texto '''
chars = sorted(list(set(text)))
print('Total de caracteres:', len(chars))
char_indices = dict((c, i) for i, c in enumerate(chars))
indices_char = dict((i, c) for i, c in enumerate(chars))
maxlen = 40

''' Define a funcao que seleciona um caractere de maneira probabilistica '''
def sample(preds, temperature=1.0):
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds) / temperature
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)
    probas = np.random.multinomial(1, preds, 1)
    return np.argmax(probas)

''' Carrega o modelo de rede neural que ja foi treinado '''
model = load_model('modelo_biblia_160.h5')

''' Produz um fragmento de texto de 200 caracteres '''
```

```
for iteration in range(1,200):
    print()
    print('-' * 50)
    start_index = random.randint(0, len(text) - maxlen - 1)
    print()
    generated = ''
    sentence = text[start_index: start_index + maxlen]
    generated += sentence
    print('----- Generating with seed: "' + sentence + '"')
    sys.stdout.write(generated)

    for i in range(160):
        x = np.zeros((1, maxlen, len(chars)))
        for t, char in enumerate(sentence):
            x[0, t, char_indices[char]] = 1.

        preds = model.predict(x, verbose=0)[0]
        next_index = sample(preds)
        next_char = indices_char[next_index]

        generated += next_char
        sentence = sentence[1:] + next_char

        sys.stdout.write(next_char)
        sys.stdout.flush()
    print()
```

APÊNDICE F – CÓDIGO DE TREINAMENTO DA TERCEIRA REDE

```
from __future__ import print_function
from keras.models import Sequential, load_model
from keras.layers import Dense, Activation
from keras.layers import LSTM, Dropout
from keras.optimizers import RMSprop
from keras.utils.data_utils import get_file
import numpy as np
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error

''' Carrega o banco de dados '''
dataset = read_csv('pollution.csv', header=0, index_col=0)
values = dataset.values
encoder = LabelEncoder()
values = values[:,(0,1,2,3,5,6,7)]
values = values.astype('float32')

''' Transforma os dados para valores de 0 a 1 '''
scaler = MinMaxScaler(feature_range=(0, 1))
values = scaler.fit_transform(values)
train = values[:, :]

''' Divide os dados em diversas sequencias de comprimento 72 '''
maxlen = 72
step = 1
train_X = []
train_Y = []
inicial = []
for i in range(0, len(train) - maxlen, step):
```

```
    train_X.append(train[i: i + maxlen, :])
    train_Y.append(train[i + maxlen, :])
    inicial.append(i)
print('Total de sequencias:', len(train_X))

''' Transforma cada uma das sequencias em matrizes no formato adequado '''
X = np.zeros((len(train_X), maxlen, len(train_X[0][0])))
y = np.zeros((len(train_X), len(train_X[0][0])))
for i, sequence in enumerate(train_X):
    for t, dados in enumerate(sequence):
        for dado, valor in enumerate(dados):
            X[i, t, dado] = train_X[i][t][dado]
            y[i, dado] =  train_Y[i][dado]

''' Cria o modelo de rede neural com 2 camadas LSTM alternadas com 2
camadas Dropout '''
model = Sequential()
model.add(LSTM(256, return_sequences=True, input_shape=(maxlen,
len(train_X[0][0]))))
model.add(Dropout(0.2))
model.add(LSTM(256))
model.add(Dropout(0.2))
model.add(Dense(len(train_X[0][0])))
model.compile(loss='mae', metrics=['mse'], optimizer='adam')

''' Realiza o treinamento da rede, salvando os modelos ao fim de cada
iteracao '''
for iteration in range(1, 200):
    print()
    print('-' * 50)
    print('Iteracao', iteration)
    model.fit(X, y, batch_size=1024, epochs=1)
    model.save('modelo_pollution_'+str(iteration)+'.h5')
```

APÊNDICE G – CÓDIGO PARA TESTE DO TERCEIRO MODELO

```
from __future__ import print_function
from keras.models import Sequential, load_model
from keras.layers import Dense, Activation
from keras.layers import LSTM
from keras.optimizers import RMSprop
from keras.utils.data_utils import get_file
import numpy as np
import random
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
from time import sleep

''' Carrega o banco de dados '''
dataset = read_csv('pollution.csv', header=0, index_col=0)
values = dataset.values
encoder = LabelEncoder()
values = values[:,(0,1,2,3,5,6,7)]
values = values.astype('float32')

''' Transforma os dados para valores de 0 a 1 '''
scaler = MinMaxScaler(feature_range=(0, 1))
values = scaler.fit_transform(values)
train = values[:, :]

''' Divide os dados em diversas sequencias de comprimento 72 '''
maxlen = 72
step = 1
train_X = []
train_Y = []
```

```
inicial = []
for i in range(0, len(train) - maxlen, step):
    train_X.append(train[i: i + maxlen, :])
    train_Y.append(train[i + maxlen, :])
    inicial.append(i)

''' Carrega o modelo de rede neural que ja foi treinado '''
model=load_model('modelo_pollution_103.h5')

''' Realiza a previsao de dois dias de dados '''
for iteration in range(1,100):
            start_index = random.randint(0, len(train_X))
            print()
            sentence = train_X[start_index].copy()
            previsao = train_X[start_index].copy()
            correto = train_X[start_index].copy()

            for k in range(48):
                x = sentence.reshape(1,maxlen,7)
                preds = model.predict(x, verbose=0)
                for i in range(len(preds)):
                    if preds[0,i]>1:
                        preds[0,i]=1
                    elif preds[0,i]<0:
                        preds[0,i]=0
                for i in range(0,maxlen-1):
                    for j in range(0,7):
                        sentence[i,j] = x[0,i+1,j]
                certo = values[inicial[start_index]+maxlen+k].reshape(1,7)
                sentence[-1]=preds[0,:]
                previsao = concatenate((previsao, preds), axis=0)
                correto = concatenate((correto, certo), axis=0)

            ''' Apresenta os graficos da previsao da rede e os dados
            corretos '''
            groups = [0, 1, 2, 3, 4, 5, 6]
            colunas = ['Poluicao', 'Orvalho', 'Temperatura', 'Pressao',
            'Vento', 'Neve', 'Chuva']
            pyplot.figure()
```

```
correto = scaler.inverse_transform(correto)
previsao = scaler.inverse_transform(previsao)

for group in groups:
    pyplot.subplot(len(groups), 1, group+1)
    pyplot.plot(correto[:, group], 'b-', previsao[:, group], 'r-')
    pyplot.title(colunas[group], y=0.5, loc='right')
pyplot.show()
```

APÊNDICE H – CÓDIGO DE PROCESSAMENTO DOS DADOS PARA REALIZAÇÃO DE MOVIMENTOS NO PONG

```
import pygame
from pygame.locals import *
from sys import exit
import random
import time
import pickle

''' Define a largura e a altura da tela '''
WIDTH = 640
HEIGHT = 480

''' Define a velocidade da bola e da barra '''
speed_bola = 250.
speed_barra = 350.

''' Define uma classe para representar a bola '''
class Bola:
    def __init__(self, x, y, arq):
        self.x = x
        self.y = y
        ''' Carrega o arquivo da bola e os valores de altura e largura '''
        self.img = pygame.image.load(arq).convert_alpha()
        self.w = self.img.get_width()
        self.h = self.img.get_height()
        self.vx=speed_bola
        self.vy=speed_bola

    ''' Atualiza a posicao da bola '''
    def update(self, dt=1.):
        self.x = self.x + self.vx*dt
        self.y = self.y + self.vy*dt

        ''' Verifica se a bola bateu em alguma das barras, alterando a sua
        velocidade em x '''
        if self.x <= 10. + barra1.w:
```

```
            if self.y >= barra1.y - self.h/2. and self.y <= barra1.y +
            barra1.h - self.h/2.:
                self.x = 10. + barra1.w
                self.vx = -self.vx
        if self.x >= WIDTH - 10. - barra2.w - self.w:
            if self.y >= barra2.y - self.h/2. and self.y <= barra2.y +
            barra2.h - self.h/2.:
                self.x = WIDTH - 10. - barra2.w - self.w
                self.vx = -self.vx

        ''' Verifica se algum dos lados fez gol, fazendo com que a bola
        e as barras retornem a suas posicoes iniciais, alterando a
        direcao da bola '''
        if self.x < 5.:
            barra2.score += 1
            self.x, self.y = WIDTH/2. - self.w/2., HEIGHT/2. - self.h/2.
            barra1.y,barra2.y = HEIGHT/2. - barra1.h/2., HEIGHT/2. - barra2.h/2.
            self.vx = -self.vx
            self.vy = 250
        elif self.x > WIDTH - 5 - self.w:
            barra1.score += 1
            self.x, self.y = WIDTH/2. - self.w/2., HEIGHT/2. - self.h/2.
            barra1.y,barra2.y = HEIGHT/2. - barra1.h/2., HEIGHT/2. - barra2.h/2.
            self.vx = -self.vx
            self.vy = 250

        ''' Verifica se a bola bateu na parte superior ou inferior da tela,
        alterando sua velocidade em y '''
        if self.y <= 7.:
            self.vy = -self.vy
            self.y = 7.
        elif self.y >= HEIGHT-5.-self.h:
            self.vy = -self.vy
            self.y = HEIGHT-5.-self.h

    ''' Desenha a bola na tela '''
    def desenha(self, tela):
        x = self.x
        y = self.y
```

```
        tela.blit(self.img,(x,y))

''' Define uma classe para representar a barra '''
class Barra:
    def __init__(self, x, arq):
        self.x = x
        ''' Carrega o arquivo da barra e os valores de altura e largura '''
        self.img = pygame.image.load(arq).convert_alpha()
        self.w = self.img.get_width()
        self.h = self.img.get_height()
        self.y = HEIGHT/2. - self.h/2.
        self.v = 0.
        self.score = 0

    ''' Atualiza a posicao da bola '''
    def update(self, dt = 1.):
    self.y = self.y+self.v*dt
    ''' Define os limites superior e inferior de movimentacao da barra '''
        if self.y<7.:
            self.y = 7.
        if self.y>HEIGHT-5.-self.h:
            self.y=HEIGHT-5.-self.h

    ''' Desenha a barra na tela '''
    def desenha(self, tela):
        x = self.x
        y = self.y
        tela.blit(self.img,(x,y))

''' Inicializa o pygame, criando uma tela de jogo '''
pygame.init()
tela = pygame.display.set_mode((WIDTH,HEIGHT))
''' Cria um plano de fundo todo branco para ser aplicado inicialmente apos
cada update da tela '''
back = pygame.Surface((WIDTH,HEIGHT))
background = back.convert()
background.fill((255,255,255))

''' Cria as barras, uma em cada canto da tela, e a bola no centro '''
```

```
barra1 = Barra(10.,'Barra.png')
barra2 = Barra(WIDTH-20.,'Barra.png')
bola = Bola(WIDTH/2.-7.5,HEIGHT/2.-7.5,'bolinha.png')

''' Define as fontes de escria do placar e do tempo '''
font = pygame.font.SysFont("calibri",40)
font2 = pygame.font.SysFont("calibri",20)

''' Define o clock que sera usado para mudar a taxa de atualizacao da tela
e dos objetos e o tempo inicial '''
clock = pygame.time.Clock()
ini = time.time()

''' Inicializa o banco de dados como um conjunto vazio '''
dados = []

''' Roda o programa por 20 minutos, gravando as informacoes desejadas '''
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()
        ''' Verifica se algum dos botoes w,s, UP ou DOWN esta pressionado,
        alterando a velocidade da barra '''
        if event.type == KEYDOWN:
            if event.key == K_w:
                barra1.v = -speed_barra
            elif event.key == K_s:
                barra1.v = speed_barra
            if event.key == K_UP:
                barra2.v = -speed_barra
            elif event.key == K_DOWN:
                barra2.v = speed_barra
        ''' Verifica se algum dos botoes w,s, UP ou DOWN deixou de ser
        pressionado, fazendo a barra parar '''
        if event.type == KEYUP:
            if event.key == K_UP:
                barra2.v = 0
            elif event.key == K_DOWN:
                barra2.v = 0
```

```
        if event.key == K_w:
            barra1.v = 0
        elif event.key == K_s:
            barra1.v = 0

''' Conta o tempo que passou desde o ultimo update (em ms), considerando
uma frequencia de 60Hz '''
time_passed = clock.tick(60)
time_passed = 1000/60
time_sec = time_passed / 1000.0

''' Atualiza o tempo e as posicoes, placares e velocidades dos objetos '''
bola.update(time_sec)
barra1.update(time_sec)
barra2.update(time_sec)
atual = time.time()
temp = atual-ini

''' Salva os placares e o tempo para serem mostrados na tela '''
score1 = font.render(str(barra1.score), True,(0,0,0))
score2 = font.render(str(barra2.score), True,(0,0,0))
tempo = font2.render(str(round(temp,2)), True,(0,0,0))

''' Desenha o fundo branco na tela, pinta o interior da quadra, o meio
campo e o contorno '''
tela.blit(background,(0,0))
quadra = pygame.draw.rect(tela,(200,200,255),Rect((5,5),
(WIDTH-10.,HEIGHT-10.)))
contorno = pygame.draw.rect(tela,(0,0,0),Rect((5,5),
(WIDTH-10.,HEIGHT-10.)), 2)
meio_campo = pygame.draw.aaline(tela,(0,0,0),(WIDTH/2.,5.),
(WIDTH/2.,HEIGHT-5.))

''' Desenha os objetos, os placares e o tempo na tela '''
bola.desenha(tela)
barra1.desenha(tela)
barra2.desenha(tela)
tela.blit(score1,(WIDTH/2.-80.,HEIGHT/2.-30.))
tela.blit(score2,(WIDTH/2.+50.,HEIGHT/2.-30.))
```

```
    tela.blit(tempo,(WIDTH/2.-20,20))

    ''' Adiciona as informacoes atuais aos dados '''
    dados.append([barra1.v,barra2.v,barra1.y,barra2.y,bola.x,bola.vx,bola.y,
    bola.vy])

    ''' Atualiza a tela, mostrando os objetos adicionados '''
    pygame.display.update()

    ''' Encerra o programa e salva os dados ao atingir 20 minutos '''
    if len(dados)==72000:
        break

pickle.dump( dados, open( "pong.p", "wb" ))
```

APÊNDICE I – CÓDIGO DE TREINAMENTO DA QUARTA REDE

```
from __future__ import print_function
from keras.models import Sequential, load_model
from keras.layers import Dense, Activation
from keras.layers import LSTM, Dropout
from keras.optimizers import RMSprop
from keras.utils.data_utils import get_file
import numpy as np
import random
import sys
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
import pickle

''' Carrega o banco de dados e transforma os dados em valores de 0 a 1 '''
values = pickle.load( open( "pong.p", "rb" ) )
scaler = MinMaxScaler(feature_range=(0, 1))
values = scaler.fit_transform(values)
train = values[:, :]

''' Divide os dados em diversas sequencias de comprimento 120 '''
maxlen = 120
step = 1
train_X = []
train_Y = []
inicial = []
for i in range(0, len(train) - maxlen, step):
    train_X.append(train[i: i + maxlen, :])
    train_Y.append(train[i + maxlen, :])
    inicial.append(i)
```

```
print('Total de sequencias:', len(train_X))

''' Transforma cada uma das sequencias em matrizes no formato adequado '''
X = np.zeros((len(train_X), maxlen, len(train_X[0][0])))
y = np.zeros((len(train_X), len(train_X[0][0])))
for i, sequence in enumerate(train_X):
    for t, dados in enumerate(sequence):
        for dado, valor in enumerate(dados):
            X[i, t, dado] = train_X[i][t][dado]
            y[i, dado] =  train_Y[i][dado]


''' Cria o modelo de rede neural com 2 camadas LSTM alternadas com 2 camadas
Dropout '''
print('Build model...')

model = Sequential()
model.add(LSTM(256, return_sequences=True, input_shape=(maxlen,
len(train_X[0][0]))))
model.add(Dropout(0.2))
model.add(LSTM(256))
model.add(Dropout(0.2))
model.add(Dense(len(train_X[0][0])))
model.compile(loss='mae', metrics = ['mse'], optimizer='adam')

''' Realiza o treinamento da rede, salvando os modelos ao fim de cada
iteracao '''
for iteration in range(1, 200):
    print()
    print('-' * 50)
    print('Iteration', iteration)
    model.fit(X, y, batch_size=1024, epochs=1)
    model.save('modelo_pong_'+str(iteration)+'.h5')
```

APÊNDICE J – CÓDIGO PARA TESTE DO QUARTO MODELO

```
import pygame
from keras.models import load_model
from sklearn.preprocessing import MinMaxScaler
from pygame.locals import *
from sys import exit
import random
import time
import pickle

''' Carrega o banco de dados e transforma os dados em valores de 0 a 1 '''
values = pickle.load( open( "pong.p", "rb" ) )
scaler = MinMaxScaler(feature_range=(0, 1))
values = scaler.fit_transform(values)

''' Define a largura e a altura da tela '''
WIDTH = 640
HEIGHT = 480

''' Define a velocidade da bola e da barra '''
speed_bola = 250.
speed_barra = 350.

''' Define uma classe para representar a bola '''
class Bola:
    def __init__(self, x, y, arq):
        self.x = x
        self.y = y
        ''' Carrega o arquivo da bola e os valores de altura e largura '''
        self.img = pygame.image.load(arq).convert_alpha()
        self.w = self.img.get_width()
        self.h = self.img.get_height()
        self.vx=speed_bola
        self.vy=speed_bola

    ''' Atualiza a posicao da bola '''
    def update(self, dt=1.):
```

```
self.x = self.x + self.vx*dt
self.y = self.y + self.vy*dt

''' Verifica se a bola bateu em alguma das barras, alterando a sua
velocidade em x '''
if self.x <= 10. + barra1.w:
    if self.y >= barra1.y - self.h/2. and self.y <= barra1.y +
    barra1.h - self.h/2.:
        self.x = 10. + barra1.w
        self.vx = -self.vx
if self.x >= WIDTH - 10. - barra2.w - self.w:
    if self.y >= barra2.y - self.h/2. and self.y <= barra2.y +
    barra2.h - self.h/2.:
        self.x = WIDTH - 10. - barra2.w - self.w
        self.vx = -self.vx

''' Verifica se algum dos lados fez gol, fazendo com que a bola
e as barras retornem a suas posicoes iniciais, alterando a
direcao da bola '''
if self.x < 5.:
    barra2.score += 1
    self.x, self.y = WIDTH/2. - self.w/2., HEIGHT/2. - self.h/2.
    barra1.y,barra2.y = HEIGHT/2. - barra1.h/2., HEIGHT/2. - barra2.h/2.
    self.vx = -self.vx
    self.vy = 250
elif self.x > WIDTH - 5 - self.w:
    barra1.score += 1
    self.x, self.y = WIDTH/2. - self.w/2., HEIGHT/2. - self.h/2.
    barra1.y,barra2.y = HEIGHT/2. - barra1.h/2., HEIGHT/2. - barra2.h/2.
    self.vx = -self.vx
    self.vy = 250

''' Verifica se a bola bateu na parte superior ou inferior da tela,
alterando sua velocidade em y '''
if self.y <= 7.:
    self.vy = -self.vy
    self.y = 7.
elif self.y >= HEIGHT-5.-self.h:
    self.vy = -self.vy
```

```
            self.y = HEIGHT-5.-self.h

    ''' Desenha a bola na tela '''
    def desenha(self, tela):
        x = self.x
        y = self.y
        tela.blit(self.img,(x,y))

''' Define uma classe para representar a barra '''
class Barra:
    def __init__(self, x, arq):
        self.x = x
        ''' Carrega o arquivo da barra e os valores de altura e largura '''
        self.img = pygame.image.load(arq).convert_alpha()
        self.w = self.img.get_width()
        self.h = self.img.get_height()
        self.y = HEIGHT/2. - self.h/2.
        self.v = 0.
        self.score = 0

    ''' Atualiza a posicao da bola '''
    def update(self, dt = 1.):
    self.y = self.y+self.v*dt
    ''' Define os limites superior e inferior de movimentacao da barra '''
        if self.y<7.:
            self.y = 7.
        if self.y>HEIGHT-5.-self.h:
            self.y=HEIGHT-5.-self.h

    ''' Desenha a barra na tela '''
    def desenha(self, tela):
        x = self.x
        y = self.y
        tela.blit(self.img,(x,y))

''' Carrega o modelo de rede neural que ja foi treinado '''
model=load_model('modelo_pong_'+str(61)+'.h5')

''' Inicializa o pygame, criando uma tela de jogo '''
```

```
pygame.init()
tela = pygame.display.set_mode((WIDTH,HEIGHT))
''' Cria um plano de fundo todo branco para ser aplicado inicialmente apos
cada update da tela '''
back = pygame.Surface((WIDTH,HEIGHT))
background = back.convert()
background.fill((255,255,255))

''' Cria as barras, uma em cada canto da tela, e a bola no centro '''
barra1 = Barra(10.,'Barra.png')
barra2 = Barra(WIDTH-20.,'Barra.png')
bola = Bola(WIDTH/2.-7.5,HEIGHT/2.-7.5,'bolinha.png')

''' Define as fontes de escria do placar e do tempo '''
font = pygame.font.SysFont("calibri",40)
font2 = pygame.font.SysFont("calibri",20)

''' Define o clock que sera usado para mudar a taxa de atualizacao da tela
e dos objetos e o tempo inicial '''
clock = pygame.time.Clock()
ini = time.time()

''' Roda o programa por 60 segundos, prevendo os movimentos adequados '''
dados = []
numero=0
while True:
    ''' Carrega os valores usados nos 2 primeiros segundos '''
    if numero<120:
        dados.append([barra1.v,barra2.v,barra1.y,barra2.y,bola.x,bola.vx,bola.y,
        bola.vy])
        barra1.v = values[numero][0]
        barra2.v = values[numero][1]

    ''' Preve o movimento de cada barra nos instantes seguintes '''
    else:
        X = np.reshape(dados,(1,maxlen,8))
        preds = model.predict(X, verbose=0)
        for i in range(len(preds[0])):
            if preds[0,i]>1:
```

```
            preds[0,i]=1
        elif preds[0,i]<0:
            preds[0,i]=0
    for i in range(0,maxlen-1):
        for j in range(0,8):
            dados[i][j] = X[0,i+1,j]

    dados[-1]=[barra1.v,barra2.v,barra1.y,barra2.y,bola.x,bola.vx,
    bola.y,bola.vy]
    dados[-1][0]=preds[0,0]
    dados[-1][1]=preds[0,1]

    barra1.v = (2.*preds[0,0]-1.)*speed_barra
    barra2.v = (2.*preds[0,1]-1.)*speed_barra

''' Conta o tempo que passou desde o ultimo update (em ms),
considerando uma frequencia de 60Hz '''
time_passed = clock.tick(60)
time_passed = 1000/60
time_sec = time_passed / 1000.0

''' Atualiza o tempo e as posicoes, placares e velocidades dos objetos '''
bola.update(time_sec)
barra1.update(time_sec)
barra2.update(time_sec)
atual = time.time()
temp = atual-ini

''' Salva os placares e o tempo para serem mostrados na tela '''
score1 = font.render(str(barra1.score), True,(0,0,0))
score2 = font.render(str(barra2.score), True,(0,0,0))
tempo = font2.render(str(round(temp,2)), True,(0,0,0))

''' Desenha o fundo branco na tela, pinta o interior da quadra, o meio
campo e o contorno '''
tela.blit(background,(0,0))
quadra = pygame.draw.rect(tela,(200,200,255),Rect((5,5),
(WIDTH-10.,HEIGHT-10.)))
contorno = pygame.draw.rect(tela,(0,0,0),Rect((5,5),
```

```
(WIDTH-10.,HEIGHT-10.)), 2)
meio_campo = pygame.draw.aaline(tela,(0,0,0),(WIDTH/2.,5.),
(WIDTH/2.,HEIGHT-5.))

''' Desenha os objetos, os placares e o tempo na tela '''
bola.desenha(tela)
barra1.desenha(tela)
barra2.desenha(tela)
tela.blit(score1,(WIDTH/2.-80.,HEIGHT/2.-30.))
tela.blit(score2,(WIDTH/2.+50.,HEIGHT/2.-30.))
tela.blit(tempo,(WIDTH/2.-20,20))

''' Adiciona as informacoes atuais aos dados '''
dados.append([barra1.v,barra2.v,barra1.y,barra2.y,bola.x,bola.vx,bola.y,
bola.vy])

''' Atualiza a tela, mostrando os objetos adicionados '''
pygame.display.update()

''' Encerra o programa ao atingir 60s '''
if len(dados)==3600:
    break
```