Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Departamento de Engenharia Elétrica

Trabalho de Conclusão de Curso

Controlador Preditivo não Linear Embarcado para Aeropêndulo Utilizando Rede Neural Artificial

Arthur Dimitri Brito Oliveira

Campina Grande, Paraíba, Brasil

©Arthur Dimitri Brito Oliveira, 25 de maio de 2021

Arthur Dimitri Brito Oliveira

Controlador Preditivo não Linear Embarcado para Aeropêndulo Utilizando Rede Neural Artificial

Trabalho de Conclusão de Curso de Bacharelado submetido à Coordenadoria de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Orientador: Prof. Rafael Bezerra Correia Lima, Dr.

Campina Grande, Paraíba, Brasil

©Arthur Dimitri Brito Oliveira, 25 de maio de 2021

Arthur Dimitri Brito Oliveira

Controlador Preditivo não Linear Embarcado para Aeropêndulo Utilizando Rede Neural Artificial

Trabalho de Conclusão de Curso de Bacharelado submetido à Coordenadoria de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Aprovado em __/__/__

Prof. Rafael Bezerra Correia Lima, Dr.

Orientador

Prof. Péricles Rezende Barros, Ph.D

Avaliador

Campina Grande, Paraíba, Brasil

©Arthur Dimitri Brito Oliveira, 25 de maio de 2021

Agradecimentos

Muitos foram os desafios que culminaram na finalização desta etapa. Desde o início dessa jornada, no dia 06 de Abril de 2015, tive ao meu lado pessoas inesquecíveis que me mostraram que nenhum ser humano é uma ilha. Saibam que este trabalho tem parte de vocês, direta ou indiretamente, e isso me impele a agradecê-los.

Agradeço a Deus pela graça que permeia os meus dias e que me trouxe até aqui, e pela compreensão de que não há um centímetro quadrado sequer da existência onde não prevaleça a sua soberania.

Aos meus familiares, incluindo meus pais, Agostinho e Luciana, meus tios Ranyere, Rosângela e Jandira, e aos meus avós Agostinho, Aracy (*in memorian*) e Juracy, agradeço por todo o amor e suporte ao longo da minha existência. Tenham certeza que muito do que sou tem parte de vocês.

A minha namorada Débora, minha gratidão pelos incentivos, amor e presença que me fizeram encontrar tranquilidade nos momentos mais turbulentos. Aos meus sogros, Flávio e Adelle, obrigado pelo suporte, disponibilidade e carinho ao longo destas últimas semanas que não foram fáceis.

Agradeço ao professor Rafael pela disponibilidade e paciência na orientação neste trabalho. Ao professor Eanes, o meu muito obrigado pela oportunidade concedida na orientação do PIBIT, pela paciência e assistência, e por me transmitir sua paixão por IA. A Stayner meus agradecimentos pelo auxílio nos experimentos que resultaram na publicação do artigo relacionado a este trabalho.

Aos meus amigos do eterno 15.1 e agregados: Matheus, Palhares, Camila, Fabrícia, Luan, Iago, Laís, Johaynne, Pedro Ximenes, André, Tavares, Saulo, Rabello, José Lucas, Kalil, Breno e a todos os outros que não caberiam aqui, obrigado pela companhia e amizade ao longo dessa jornada. É gratificante ter pessoas como vocês por perto.

Resumo

Este trabalho apresenta um procedimento de síntese de um controlador NMPC (*Nonlinear Model Predictive Control*) para um aeropêndulo utilizando redes neurais artificiais. Os dados, obtidos a partir da simulação do sistema e controlador NMPC em malha-fechada para um conjunto de valores de saída desejados, são usados para treinamento de uma rede neural artificial. O controlador emulado é incorporado como controlador da planta e obtém um desempenho similar no rastreamento de referência. Esta abordagem reduz o custo computacional envolvido no controle NMPC original, tornando possível a execução um controlador NMPC sintetizado em um *hardware* com capacidade de processamento limitada.

Palavras-chave: Controle preditivo não linear, redes neurais artificiais, sistemas embarcados.

Abstract

This work proposes a synthesis procedure of an NMPC (Nonlinear Model Predictive Control) using neural networks for an aero-pendulum. The data, obtained by simulating the closed-loop response of the system and an NMPC controller for a range of desired values to the system output, is used to train an artificial neural network. The emulated controller is used to control the plant and achieves similar reference-tracking performance. This approach significantly reduces the computational burden involved in the NMPC original approach, making it possible to run a synthetized NMPC controller on low processing capability hardware.

Keywords: Nonlinear predictive control, artificial neural networks, embedded systems.

Sumário

Lista de símbolos e abreviaturas

LSTM Long Short Term Memory 43

MLP Multi Layered Perceptron 43

MPC Model Predictive Control 43

MPC Nonlinear Model Predictive Control 43

NLP Nonlinear Programming 46

NLP Nonlinear Programming 43

ReLU Rectified Linear Unit 43

SQP Sequential Quadratic Programming 43

UFCG Universidade Federal de Campina Grande 43

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Este capítulo destina-se à apresentação da motivação associada ao desenvolvimento deste trabalho. Ao final, há uma breve descrição das seções presentes neste documento.

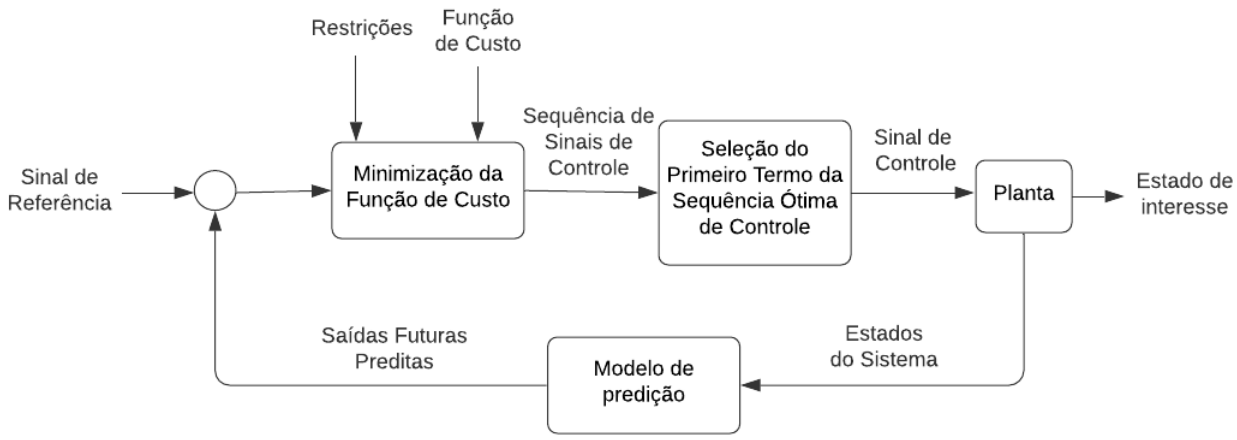
1.1 Justificativa

A técnica de controle ótimo está associada ao projeto de um controlador que conduz o sistema à estabilidade em malha fechada diante de restrições impostas às variáveis do processo, mediante minimização de uma função de custo associada [1]. Na literatura, este tipo de controle também pode ser denominado de *Moving Horizon Control*, ou Controle de Horizonte Móvel, onde a resolução do problema de controle se dá de forma ótima [2]. O Controle MPC (*Model Predictive Control*) é uma das técnicas de controle capazes de solucionar este tipo de problema.

A nomenclatura MPC está associada ao conceito de utilizar um modelo da planta a ser controlada para predizer a evolução temporal dos estados. Levando-se em consideração um conjunto pré-definido de restrições, utiliza-se um modelo dinâmico do processo para estimar a sua evolução temporal e calcula-se a ação de controle mais adequada [3] para o instante atual. Dentro de uma janela de tempo limitada, minimiza-se o custo associado a uma função alvo. Este custo está represente a diferença entre a saída predita pelo controlador e o sinal de referência desejado. No caso de sistemas lineares, normalmente recorre-se à utilização de métodos de programação quadrática para resolução do problema de otimização. O diagrama

ilustrativo do algoritmo de MPC pode ser visualizado na Figura 1.1.

Figura 1.1: Diagrama de blocos de um sistema com controle MPC.



Fonte: autoria própria.

No método MPC, portanto, a cada instante de amostragem resolve-se um problema de controle ótimo. Uma sequência de ações de controle é calculada e aplica-se o primeiro termo desta sequência no processo. A necessidade de resolução de um único problema de controle, de forma exclusiva para o estado atual do sistema, diminui consideravelmente a complexidade do procedimento.

A utilização de MPC pode ser reforçada por inúmeras vantagens sobre os métodos de controle tradicionais, como a capacidade de lidar com processos multivariáveis, sistemas com atraso e a rejeição a perturbações. Além disso, é um método capaz de lidar com restrições impostas às entradas e saídas do sistema e possui inúmeras ferramentas de desenvolvimento disponíveis para utilização. No entanto, requer um modelo do sistema, o que implica em experimentações, identificação do sistema e linearização no caso de sistemas não-lineares.

A maior parte dos sistemas reais têm natureza não-linear. Há métodos bem consolidados como a linearização por *feedback* [4], [5], [6] para lidar com processos não-lineares. No entanto, tais abordagens necessitam de procedimentos de projeto complexos e, de certa forma, suprimem informações sobre a dinâmica do sistema. Quando há a necessidade de resolver um problema de controle ótimo e o sistema precisa ser caracterizado por meio de modelos não lineares, o uso de uma técnica de controle preditivo não linear torna-se atrativa.

O método MPC linear está associado a esquemas de controle MPC baseados em modelos

lineares do sistema, nos quais restrições lineares aplicadas a variáveis do processo e funções de custo quadráticas são utilizadas. NMPC (*Nonlinear Model Predictive Control)*, por sua vez, refere-se a abordagens MPC que utilizam modelos não-lineares na predição do sinal de controle, permitindo a aplicação de funções de custo não-quadráticas e restrições não-lineares sobre as variáveis do processo. Como acontece no método de MPC linear, a abordagem não-linear também requer uma solução iterativa do problema de controle ótimo dentro de um horizonte de predição finito. No entanto, o custo computacional associado ao NMPC é maior [7] devido à possibilidade de resolução de problemas de otimização envolvendo regiões não-convexas.

Sistemas embarcados necessitam de algoritmos de controle seguros e capazes de operar diante de limitações de memória e processamento. Algoritmos NMPC satisfazem restrições operacionais, embora sejam desafiadores no que diz respeito à implementação em hardware quando a velocidade de operação em tempo real requerida é elevada [8]. Além disso, deve-se levar em consideração o pior caso de tempo de execução. Nesse contexto, a depender das capacidades computacionais associadas à resolução dos problemas de otimização, pode ser necessário um deslocamento do esforço computacional para uma unidade de processamento mais robusta, utilizando, por exemplo, um protocolo de comunicação OPC [9] para envio dos dados de controle à planta. Todavia, caso haja uma incompatibilidade entre a velocidade de dinâmica do sistema e o processamento/envio dos dados, este tipo de solução pode incorporar obstáculos temporais ao sistema de controle.

Redes neurais artificiais podem ser definidas como aproximadores gerais de funções [10]. O teorema da aproximação universal afirma que uma rede neural artificial com uma única camada oculta, contendo um número finito de neurônios, pode aproximar qualquer função contínua, dados pressupostos mínimos quanto às funções de ativação [11]. Diante da complexidade envolvida na execução em tempo real dos métodos de otimização presentes nos algoritmos MPC, uma alternativa é utilizar redes neurais para emular o comportamento de controladores preditivos. No contexto da aprendizagem supervisionada aplicada a problemas de controle, os estados do sistema são fornecidos à rede, bem como os sinais de controle desejados. O objetivo consiste na minimização da função de custo, que expressa o quanto distam os sinais de controle calculados pela rede, a partir das entradas, dos sinais de controle originalmente aplicados pelo controlador.

Atualmente, algumas abordagens utilizam redes MLP (*Multilayer Perceptron*) como método de síntese para controladores preditivos [12], [13] por meio de treinamentos supervisionados. Redes do tipo LSTM (*Long Short Term Memory*) também têm sido utilizadas para este tipo de emulação [14], dada a capacidade destas arquiteturas de levar em consideração os comportamentos passados e presentes do sistema para a predição dos valores futuros.

1.2 Estudo de Caso

Neste trabalho, propõe-se a aplicação de um controlador neural embarcado baseado em NMPC. A planta utilizada para avaliação da metodologia foi um aeropêndulo, sistema amplamente difundido na literatura [15], [16]. Adotou-se o MATLAB/Simulink como ferramenta de simulação e controle do sistema. Essa escolha foi feita devido à *toolbox* de NMPC existente no *software*.

O estudo consistiu na simulação do sistema em malha fechada com um controlador NMPC convencional, utilizando um conjunto aleatório de referências para um dos estados. Os dados do referido experimento foram coletados, incluindo as referências adotadas, os valores dos estados e os sinais de controle aplicados à planta. Estes dados foram separados em conjuntos de treino e teste, sendo utilizados para treinamento e validação de desempenho de redes neurais do tipo LSTM e MLP. Subsequentemente, o controlador neural, que sintetiza a lei de controle original, foi incorporado ao microcontrolador do sistema real.

1.3 Objetivos Gerais

Este trabalho tem como objetivo propor e avaliar uma abordagem que minimize o esforço computacional associado ao algoritmo de NMPC. Mediante síntese do controlador por meio de uma rede neural artificial, enseja-se gerar um controlador com menor custo computacional e desempenho equivalente ao NMPC original, sendo capaz de ser implementado em um *hardware* de capacidades limitadas.

1.4 Objetivos Específicos

Como objetivos específicos, propõe-se a semelhança no desempenho em malha fechada para os estados de interesse, tanto na simulação, quanto no sistema real. Deseja-se avaliar, comparativamente, o desempenho em malha-fechada de arquiteturas LSTM e MLP. Além disso, há a pretensão de que o controlador neural sintetizado tenha um tempo de execução inferior ao NMPC original, com erro semelhante no problema de rastreamento de referência.

1.5 Estrutura do trabalho

O capítulo 2 trata da formulação matemática envolvida no algoritmo de NMPC e aborda os fundamentos conceituais e matemáticos relativos à aprendizagem de máquina. Posteriormente, tem-se o capítulo 3, destinado à descrição do processo e do controlador utilizado, bem como o detalhamento das experimentações realizadas e seus respectivos resultados. Por fim, há o capítulo 4, destinado às considerações finais do trabalho e à proposição de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este capítulo destina-se à conceituação do controle NMPC e os fundamentos relacionados à aprendizagem de máquina.

2.1 NMPC - Controle Preditivo Não Linear Baseado em Modelo

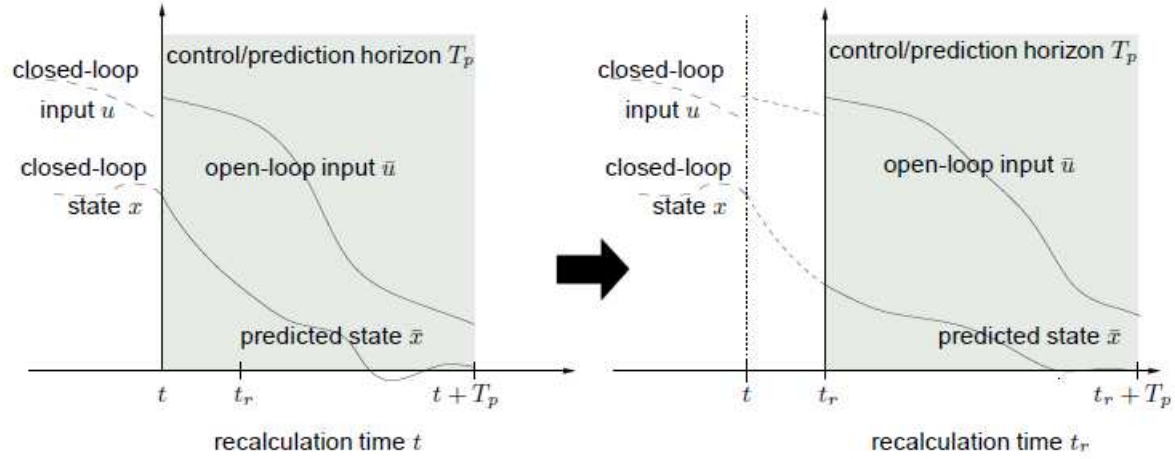
NMPC é um método baseado em otimização para o controle por realimentação de sistemas não-lineares. É utilizado majoritariamente para a resolução de problemas de estabilização e rastreamento de referência. No contexto de rastreamento, supondo um processo com estado $x(k)$, amostrado a cada instante k, pode-se selecionar entradas $u(k)$ que influenciam o seu comportamento futuro. A tarefa consiste, portanto, em determinar as entradas manipuláveis $u(k)$ para que o estado de interesse acompanhe a referência imposta $x_{ref}(k)$. Desvios de $x(k)$ em relação à sua referência $x_{ref}(k)$ devem ser corrigidos.

O objetivo consiste em um mapeamento que relacione o estado às entradas necessárias para o sistema na forma $u(k) = \mu(x(k))$. A função μ estabelece uma correspondência entre o estado e um conjunto de sinais de controle U. A partir da geração de uma sequência de entradas manipuláveis dentro de um horizonte de controle, otimiza-se uma função objetivo dentro de um horizonte de predição [17].

Na Figura 2.1, ilustra-se o princípio que rege o NMPC. O instante de amostragem t marca o início do processo de otimização. Um conjunto admissível de entradas ótimas é

determinado para a janela de tempo T_p, de tal forma que a função objetivo seja minimizada. O estado de interesse x é estimado ao aplicar a sequência de sinais de controle ao modelo não linear do processo.

Figura 2.1: Princípio de funcionamento do algoritmo de NMPC.



Fonte: FINDEISEN[17]

No instante de amostragem seguinte, t_r, novas medições do estado x são feitas após a aplicação do primeiro sinal de controle à planta. O novo horizonte de controle se estende até $t_r + T_p$ e novas ações de controle ótimas são calculadas para esta janela temporal. O procedimento continua de forma cíclica com o deslocamento do horizonte de predição.

A solução NMPC ideal teria horizontes de controle e predição infinitos. No entanto, horizontes menores implicam em soluções menos custosas para o problema de otimização *online*. Tendo um horizonte de predição infinito, há a garantia de que o sistema converge para o ponto de operação desejado mediante a aplicação da sequência de ações de controle calculada. No entanto, ao utilizar horizontes de controle finitos, existe uma diferença entre os estados preditos pelo controlador e os seus valores reais. Esta diferença é corrigida, gradualmente, mediante o recálculo da ação de controle a cada instante t_r.

2.1.1 NMPC - Descrição Matemática

A Equação 2.1 define a função objetivo de controle com relação a um modelo em tempo contínuo, computado em uma janela de tempo T_f, conforme [18].

$$minJ = \int_0^{T_f} \varphi(x(t), u(t), t)dt \tag{2.1}$$

Dentro do intervalo de tempo $[0, T_f]$, a função de custo φ é minimizada. As restrições de igualdade são expressas na Equação Equação 2.2.

$$\frac{dx(t)}{dt} = f(x(t), u(t), t) \tag{2.2}$$

A Equação 2.2 expressa as restrições de igualdade que descrevem as dinâmicas associadas ao processo não linear, expressas por um conjunto de equações diferenciais. As restrições aplicadas às variáveis podem ser encontradas na Equação 2.3 e Equação 2.4.

$$u_{LB} \leq u(t) \leq u_{UB} \tag{2.3}$$

$$x_{LB} \leq x(t) \leq x_{UB} \tag{2.4}$$

Ou seja, dentro do horizonte $[0, T_f]$, delimita-se os valores inferior e superior para as variáveis manipuláveis, bem como os limites para as variáveis de estado.

O problema de controle ótimo formulado anteriormente pode ser descrito como um método de otimização dinâmico com restrições e resolvido por meio de NLP (*Nonlinear Programming).* Nesse caso, o problema é reduzido a um problema NLP de dimensões finitas mediante discretização do sinal de entrada $u(t)$. A cada instante de amostragem Δt, o sinal de controle é considerado constante. O problema reformulado conduz à Equação 2.5. Esta representa uma função objetivo que minimiza o custo φ e é resolvida para um número finito de sinais de controle ótimos do instante $k = 1$ a p.

$$\min_{U_k} J(x(t), U_k) = \sum_{k=1}^{p} \varphi(x(t), U_k) \tag{2.5}$$

A função de custo, na forma discreta, pode ser descrita pela Equação 2.6. x_k representa

a variável de estado a ser controlada. U_k representa o sinal de controle ótimo calculado no instante de tempo t_k. As matrizes Q e R são parâmetros de ajuste para adequar a variável de estado à referência imposta e ajustar o sinal de controle.

$$\varphi = \sum_{k=1}^{p}(x_k - x_{ref})Q(x_k - x_{ref}) + \sum_{k=1}^{m}(U_k - U_{k-1})^T R(U_k - U_{k-1}) \tag{2.6}$$

A minimização descrita pela Equação 2.5 está sujeita às Equações 2.7, 2.8, 2.9 e 2.10.

$$\frac{dx(t)}{dt} = f(x(t), U_k) = 0, k = 1, .., p \tag{2.7}$$

A Equação 2.7 está associada a um conjunto de restrições impostas pelas equações diferenciais que descrevem o modelo no intervalo Δt, $\Delta t \in [t_{k-1}, t_k]$, $k = 1, ..., p$

$$U_{j-1} = U - j, j = m + 1, ..., p \tag{2.8}$$

A restrição imposta na Equação 2.8 significa que o sinal de controle permanece constante até o fim do horizonte de predição.

Os limites impostos na Equação 2.9 e na Equação 2.10 valem para todo o horizonte de predição p, onde $k = 1, ..., p$.

$$U_{LB} \leq U_k \leq U_{UB}, k = 1, ..., p \tag{2.9}$$

$$x_{LB} \leq x \leq x_{UB} \tag{2.10}$$

2.2 Aprendizagem de Máquina

Uma máquina aprende a realizar um conjunto de tarefas T, levando em consideração uma métrica de desempenho P, quando P, na realização da tarefa T, melhora com a experiência [19]. Aprendizagem de máquina refere-se, portanto, a um conjunto de técnicas que permitem ensinar computadores a resolução de uma determinada tarefa. É uma ferramenta extremamente útil quando há tarefas que envolvem soluções não-analíticas e não-solucionáveis por métodos de programação explícita.

A aprendizagem está associada à aquisição da habilidade para resolução da tarefa. A maneira como se dá o aprendizado varia de acordo com a natureza dos dados e a finalidade de aplicação do modelo. Esta subseção descreve os principais conceitos envolvidos na resolução de problemas utilizando o aprendizado de máquina.

2.2.1 Regressão Linear

O algoritmo de regressão linear consiste na modelagem de uma resposta escalar a partir de variáveis manipuláveis. O modelo deve ser capaz de tomar um conjunto de valores de entrada $x \in \mathbb{R}$ e predizer um conjunto de saída $y \in \mathbb{R}$. O mapeamento do conjunto de valores de entrada para o conjunto de saída deve se dar por $f : \mathbb{R} \to \mathbb{R}$. A saída predita pode ser expressa pela Equação 2.11.

$$y = W^T X \tag{2.11}$$

W representa um conjunto de parâmetros com os quais controla-se a saída. Estes coeficientes são multiplicados às entradas e são capazes de ponderar a influência de X sobre o valor de saída y. Supondo um vetor de alvos de regressão, y_{test}, e o conjunto de valores aproximados para a saída $\hat{y_{test}}$, obtidos por meio do modelo de regressão, uma das possíveis maneiras de avaliar o desempenho deste modelo é calcular o erro médio quadrático entre os conjuntos, que pode ser expresso pela Equação 2.12.

$$MSE_{test} = \frac{1}{m}||\hat{y_{test}} - y_{test}||^2 \tag{2.12}$$

Ou seja, o erro aumenta de forma proporcional ao incremento da distância euclidiana entre as amostras. Se este erro estiver associado a um sistema de aprendizagem de máquina, deseja-se que um determinado algoritmo seja capaz produzir MSE_{test} próximo de zero. A partir de exemplos de treinamento fornecidos, a maneira de reduzir o erro é encontrar o gradiente nulo da função associada a ele [10].

$$\nabla_w \frac{1}{m}||\hat{y_{test}} - y_{test}||^2 = 0 \tag{2.13}$$

$$\nabla_w (X_{train} w - y^{train})^T (X_{train} w - y^{train}) = 0 \tag{2.14}$$

$$w = (X_{train}^T X_{train})^{-1} X_{train}^T y_{train} \tag{2.15}$$

A expressão descrita pela Equação 2.15 é conhecida como equação normal e consiste em um simples algoritmo de aprendizagem. Normalmente adiciona-se um termo de intercepção b, frequentemente denominado de *bias*. Assim, o mapeamento dos parâmetros se dá por uma função afim, como pode-se observar na Equação 2.16.

$$\hat{y} = w^T x + b \tag{2.16}$$

2.2.2 Sobreajuste e Subajuste

Em algoritmos de aprendizagem supervisionada, onde os exemplos de treinamento consistem um conjunto de entradas e seus respectivos rótulos, deve-se escolher a função hipótese, que descreve a relação entre entradas e saída, mais apropriada. Espera-se do modelo final uma boa capacidade de generalização, de tal forma que, diante de inúmeras entradas novas associadas ao domínio do problema, o sistema tenha bom desempenho na predição dos rótulos desejados. De forma iterativa, o erro associado ao conjunto de treinamento é minimizado, resultando em um valor baixo para o erro relativo ao conjunto de teste.

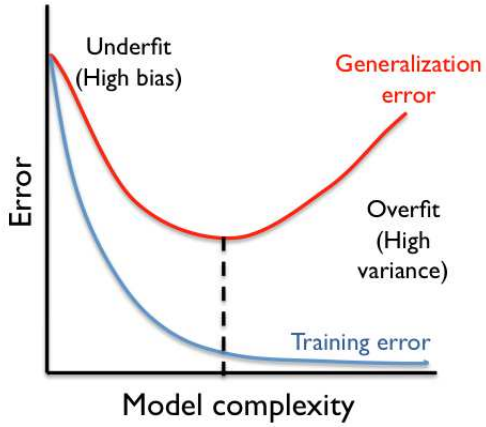
Supondo que os conjuntos de treino e teste têm a mesma distribuição de probabilidade associada, e que os exemplos em cada conjunto são independentes uns dos outros, pode-se afirmar que a geração de dados compartilha uma mesma distribuição de probabilidade p_{data} [10]. Esta afirmação permite dizer que o erro relativo ao conjunto de treino, aleatoriamente escolhido, de um determinado modelo é equivalente ao erro obtido no conjunto de teste com esse mesmo modelo. O bom desempenho de um algoritmo de aprendizagem de máquina, portanto, diz respeito à sua habilidade de reduzir o erro de treinamento e minimizar a diferença existente entre os erros de teste e treinamento.

Diz-se que o modelo está subajustado quando não há a capacidade de capturar matematicamente as nuances relativas aos dados, resultando em um taxa de erro elevada no conjunto de treinamento e em *underfitting*, subajuste aos dados. Um modelo sobreajustado é aquele que apresenta bom desempenho no conjunto de treino e um erro elevado para o

conjunto de teste. Diz-se, nesse caso, que ocorre *overfitting*, sobreajuste.

Uma relação de compromisso fica evidente. Deve haver um meio termo no que diz respeito à complexidade do modelo, de tal forma seja possível generalizar de forma satisfatória sobre dados contidos em uma mesma distribuição de probabilidade. Normalmente, o erro no conjunto de treinamento decai assintoticamente à medida em que eleva-se a complexidade do modelo, enquanto que o erro de generalização se eleva. A relação descrita pode ser observada na Figura 2.2.

Figura 2.2: Relação entre complexidade do modelo e erro.



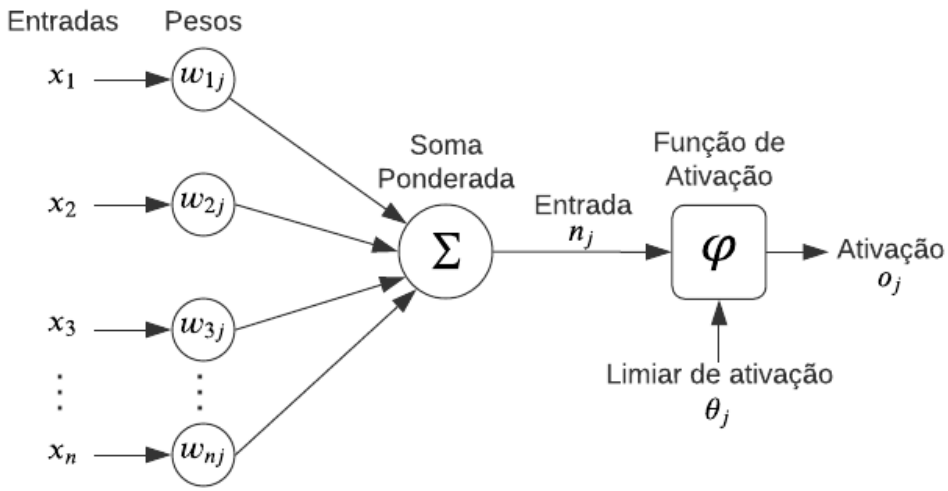
Fonte: BRANCO *et al.* [20]

Pode-se observar pela Figura 2.2 que, à medida em que aumenta-se a complexidade do modelo, naturalmente o erro no conjunto de treinamento tende a zero, já que o modelo torna-se cada vez mais ajustado aos dados. Em contrapartida, um modelo extremamente complexo resulta em um erro elevado no conjunto de testes devido à baixa capacidade de generalização. A linha tracejada indica o ponto ótimo a partir do qual não há mais benefício no incremento da complexidade do modelo.

2.2.3 Neurônio Artificial

A principal unidade de processamento de uma rede neural é denominada neurônio artificial. Dado um neurônio k, supõe-se m entradas com sinais de x_1 a x_n e pesos de w_{1j} a w_{nj}. A Figura 2.3 ilustra uma unidade neuronal.

Figura 2.3: Modelo de um neurônio artificial.



Fonte: autoria própria

A saída do neurônio o_j pode ser definida conforme exposto na Equação 2.17.

$$o_j = \varphi(\sum_{j=0}^{m} w_{kj} x_j) \tag{2.17}$$

Ou seja, a partir da multiplicação matricial entre os pesos w_kj e as entradas x_j, $j = 0, ..., m$, aplica-se uma função de ativação que transforma a computação de entrada do neurônio na saída o_j. O tipo de função aplicado às computações matriciais envolvidas pode variar conforme o problema.

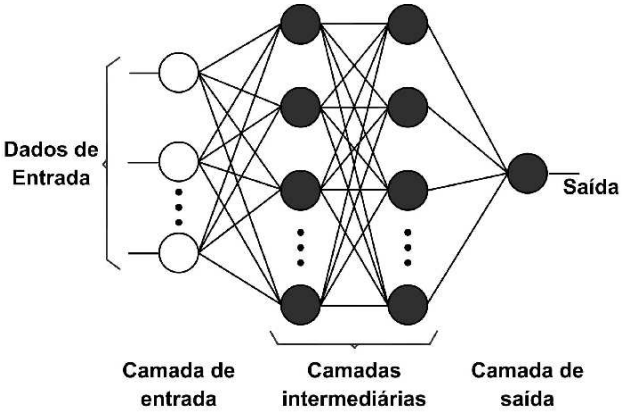
2.2.4 Redes Neurais Perceptron Multicamada

Redes neurais do tipo MLP (*Multilayer Perceptron*) são a base para o que se conhece como aprendizagem profunda. O objetivo geral de uma MLP é mapear um conjunto de entradas em uma determinada saída. Ou seja, dada uma função $y = f(x)$, o objetivo é aproximar uma função f^* que mapeie dados de entrada x em uma categoria y. Tal mapeamento pode ser expresso por meio de $y = f^*(x, w)$, onde os valores dos pesos w devem ser aprendidos para produção da saída desejada.

As redes do tipo MLP também podem ser denominadas de *feedforward*. Isso se deve pelo fluxo de informação proveniente da entrada x que sofre sucessivas computações que definem

f e geram a saída final y. O termo rede se deve ao fato de que as arquiteturas, normalmente, consistem em composições de múltiplas funções. No caso de uma rede com quatro camadas, por exemplo, as funções da primeira à quarta camada poderiam ser definidas como f_1, f_2, f_3 e f_4, sendo a saída final descrita por $y = f_4(f_3(f_2(f_1(x))))$. Um diagrama ilustrativo de uma rede MLP pode ser visualizado na Figura 2.4.

Figura 2.4: Representação gráfica de uma MLP.



Fonte: FIORIN *et al.* [21]

Uma rede neural é uma combinação de múltiplas unidades de neurônios artificiais. Cada um deles aplica simples transformações aos dados de entrada e, de forma agregada, podem resultar em composições de funções bastante complexas [10]. Supondo uma rede densamente conectada, onde cada neurônio está conectado a todos ou outros da camada anterior, as i-ésimas unidades na l-ésima camada escondida podem ser definidas como $h_i^{(l)}$. As computações podem ser descritas conforme as Equações 2.18, 2.19 e 2.20.

$$h_i^{(1)} = \phi^{(1)}(\sum_j w_{ij}^{(1)} x_j + b_i^{(1)}) \tag{2.18}$$

$$h_i^{(2)} = \phi^{(2)}(\sum_j w_{ij}^{(2)} x_j + b_i^{(2)}) \tag{2.19}$$

Há a distinção entre $\phi^{(1)}$ e $\phi^{(2)}$ nas Equações 2.18 e 2.19 devido à possibilidade de aplicação de diferentes funções e ativação em cada camada. Supondo uma rede com três camadas, a Equação 2.20 expressa a saída obtida a partir das computações realizadas na

terceira camada.

$$y = \phi^{(3)}(\sum_{j} w_{3j}^{(3)} h^{(2)} + b_i^{(3)}) \tag{2.20}$$

De forma vetorizada, as computações realizadas em cada uma das camadas podem ser expressas conforme as Equações 2.21, 2.22 e 2.23.

$$h^{(1)} = \phi^{(1)}(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \tag{2.21}$$

$$h^{(2)} = \phi^{(2)}(\mathbf{W}^{(2)}h^{(1)} + \mathbf{b}^{(2)}) \tag{2.22}$$

$$y = \phi^{(3)}(\mathbf{W}^{(3)}h^{(2)} + \mathbf{b}^{(3)}) \tag{2.23}$$

2.2.5 Funções de Ativação

O valor associado à soma ponderada das entradas de um neurônio pode variar significativamente. Funções de ativação são utilizadas em redes neurais artificiais para limitar a amplitude do sinal de saída de um neurônio por meio de uma transformação escalar-escalar. O incremento na velocidade de computação de derivadas associadas ao algoritmo de retropropagação para atualização dos pesos matriciais, bem como as propriedades matemáticas associadas à capacidade de generalização por meio do teorema geral da aproximação, podem ser apontadas como razões para a utilização de funções de ativação [22].

Uma composição de ativações lineares resulta, necessariamente, em funções lineares. Ou seja, ao utilizar-se somente ativações lineares, a complexidade associada ao modelo treinado fica limitada. Normalmente, utiliza-se uma combinação de ativações lineares e não-lineares para gerar modelos mais complexos. No caso de problemas de regressão linear, as funções de ativação mais utilizadas são a ReLU e a tanh.

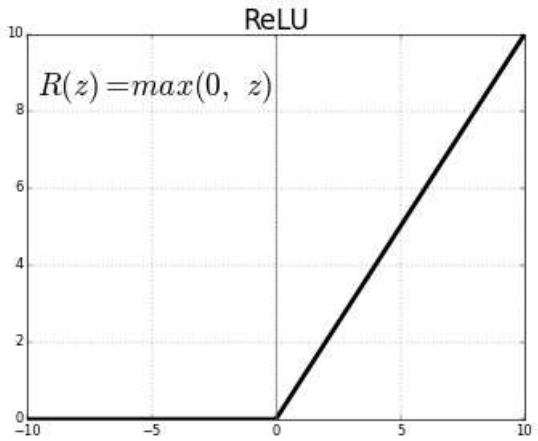
A função de ativação ReLU (*Rectified Linear Unit*) pode ser definida pela Equação 2.24.

$$f(x) = x^{+} = max(0, x) \tag{2.24}$$

A entrada do neurônio é x. A função descreve uma conjunto imagem linear para valores positivos e nulo para valores negativos. É uma função que não requer computações complexas, o que resulta em um menor tempo de treinamento ou execução. Além disso, no

contexto de aprendizagem profunda, o problema dos *vanishing gradients*, que ocorre quando há a atenuação do sinal de erro retro-propagado, é resolvido com este tipo de função de ativação. As derivadas permanecem com magnitude significativa sempre que a unidade estiver ativa. Os gradientes, além de significativos, são consistentes. A problemática deste tipo de ativação ocorre quando o valor proveniente das entradas é menor do que zero, resultando em uma derivada nula e em uma dificuldade para atualização dos parâmetros. A Figura 2.5 ilustra a ativação ReLU.

Figura 2.5: Função de ativação ReLU.

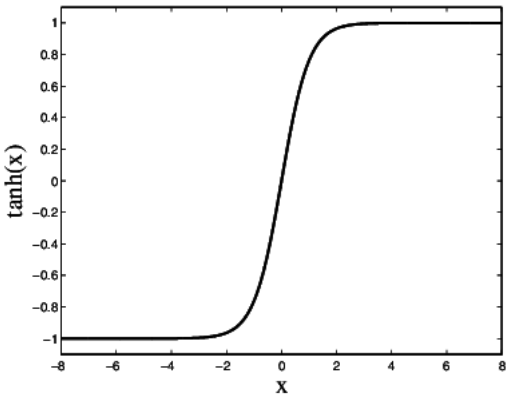


Fonte: autoria própria.

A função de ativação do tipo Tangente Hiperbólica, tanh, por sua vez, pode ser definida conforme a Equação 2.25. Ela toma quaisquer valores reais como entrada e os transforma em valores na faixa $[-1, 1]$. A ilustração gráfica dela pode ser observada na Figura 2.6.

$$tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.25}$$

Figura 2.6: Função de ativação tanh.



Fonte: autoria própria.

Uma das vantagens de se utilizar a tanh como ativação neuronal é que as entradas negativas são mapeadas como fortemente negativas e as entradas nulas são mapeadas próximas de zero no gráfico, acompanhando a variação das entradas de forma monotônica. Além disso, os gradientes têm maior magnitude em relação a outros tipos de ativações, tornando-as mais efetivas ao lidar com *vanishing gradients* em camadas intermediárias de redes *feedforward*.

2.2.6 Redes LSTM - *Long Short Term Memory*

Redes neurais MLP produzem saídas referentes a computações intermediárias a partir das entradas de forma unidirecional no caso das arquiteturas *feedforward*. Este tipo de arquitetura não é capaz de levar em consideração eventos anteriores ao mapear as relações existentes entre as entradas e as saídas para o instante atual. Quando há a necessidade de capturar relações temporais entre os eventos, arquiteturas do tipo RNN (*Recurrent Neural Network*) podem ser utilizadas.

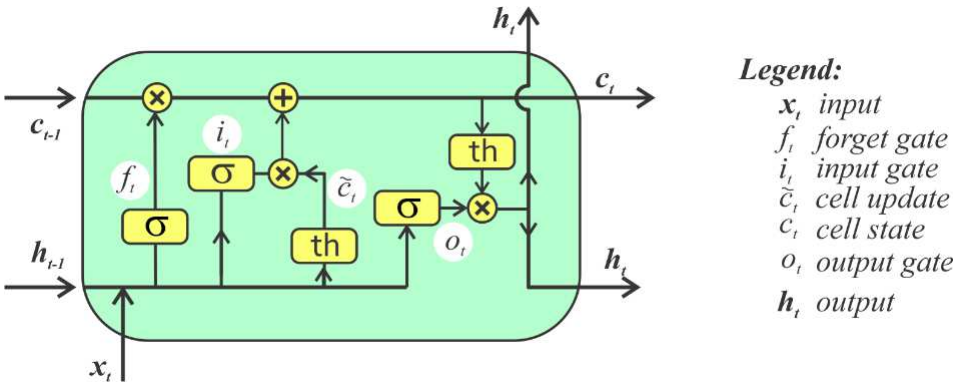
Redes recorrentes têm o fluxo de sinais em ambas as direções. Computações são derivadas da entrada a partir de realimentações, fornecendo uma espécie de memória de curto prazo que permite a composição de modelos complexos, capazes de resolver problemas com dependência temporal.

A arquitetura LSTM (*Long Short Term Memory*) é um tipo de rede neural recorrente. O seu nome está relacionado à capacidade de armazenar valores de instantes de tempo anteriores. No geral, a arquitetura permite a resolução de problemas de predição que envolvem

janelas de tempo consideráveis, generalizando de forma satisfatória e sendo imune a sinais ruidosos [23]. É uma arquitetura adequada para a resolução de problemas que envolvem séries temporais.

A célula essencial LSTM pode ser visualizada na Figura 2.7.

Figura 2.7: Unidade essencial LSTM.



Fonte: OLAH [24]

A primeira etapa executada pela célula LSTM envolve a decisão de qual informação deve ser descartada por meio de uma ativação sigmoid. A partir da saída do estado anterior h_{t-1} e da entrada atual x_t, descarta-se ou mantém-se os valores provenientes do estado c_{t-1} por meio da saída binária. A Equação 2.26 define matematicamente esta etapa.

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \tag{2.26}$$

A segunda etapa envolve a estrutura denominada de *input gate*. Inicialmente a função sigmoid filtra os valores provenientes de h_{t-1} e x_t a serem armazenados. Um vetor contendo os valores de h_{t-1} e x_t é passado por uma função tangente hiperbólica. Os valores deste vetor são multiplicados com os valores regulados, de modo a determinar informações úteis para gerar o estado da célula c_t, conforme as Equações 2.27, 2.28 e 2.29.

$$i_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \tag{2.27}$$

$$\tilde{C}_t = \tanh\left(W_c[h_{t-1}, x_t] + b_C\right) \tag{2.28}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.29}$$

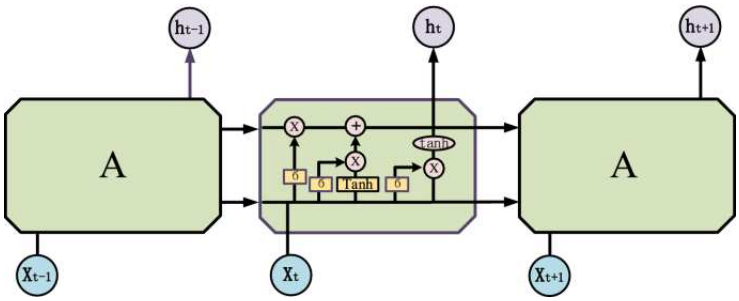
Por fim, há o *output gate*. A tarefa deste é extrair informações relevantes do estado da célula atual para compor a saída h_t. Os valores a serem armazenados, referentes às entradas h_{t-1} e x_t, passam por uma ativação do tipo sigmoid. Aplica-se uma ativação tanh ao estado da célula c_t e, posteriormente, os valores regulados são multiplicados, gerando a saída que alimentará a próxima célula conforme as Equações 2.30 e 2.31.

$$o_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \tag{2.30}$$

$$h_t = o_t * \tanh(C_t) \tag{2.31}$$

As redes LSTM são compostas pela composição em cadeia destes módulos previamente mencionados. A entradas são fornecidas aos módulos individuais e, mediante operações internas, informações relativas a instantes anteriores podem ser consideradas na predição final. A ilustração da concatenação de módulos pode ser observada na Figura 2.8.

Figura 2.8: Concatenação de módulos LSTM.



Fonte: OLAH [24]

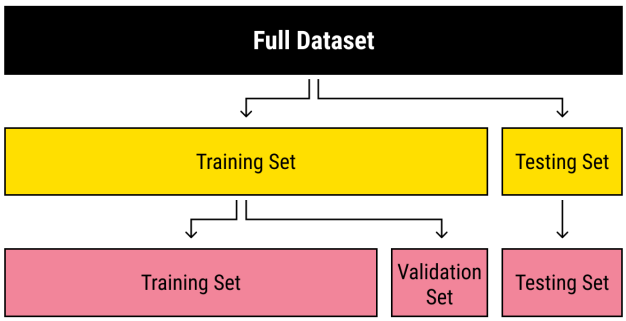
2.2.7 Treinamento de um modelo de aprendizagem de máquina

O treinamento supervisionado de um modelo de aprendizagem de máquina, mais especificamente uma rede neural, está associado ao ajuste de pesos. Nesse processo, encontra-se as matrizes que multiplicadas pelos valores de entrada resultam nos valores esperados para a saída. Por meio de exemplos, esses valores são atualizados de tal forma que a perda

associada seja minimizada.

Normalmente, no treinamento de uma modelo de aprendizagem de máquina, utiliza-se 80% dos dados para treino e 20% para teste. Os dados de teste não são utilizados no processo de treinamento, o que permite, posteriormente, classificar o modelo como sobreajustado ou não aos dados. Os dados de treinamento contém somente as variáveis de entrada, a saída é omitida e só é apresentada no momento do cálculo do custo do exemplo de treinamento. A ilustração do processo de separação do conjunto de dados pode ser observada na Figura 2.9.

Figura 2.9: Estrutura de separação do conjunto de dados.



Fonte: SYDORENKO [25]

A perda é calculada com base nas saídas da rede, sendo o resultado médio do custo relativo a cada um dos exemplos. A função de custo, por exemplo, pode ser o MSE, que para um conjunto n de exemplos pode ser definido conforme a Equação 2.32.

$$J = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2 \tag{2.32}$$

A partir do erro existente na saída, deseja-se determinar a parcela de correção aplicada a cada um dos pesos, de tal forma que o custo global seja minimizado. A técnica associada à propagação e atualização dos pesos é denominada de *backpropagation*. A partir de uma taxa de aprendizagem, os valores dos pesos são atualizados de acordo com o gradiente da perda em relação a estes pesos. Portanto, para atualizar um peso w_j com uma taxa de aprendizagem α, utiliza-se a Equação 2.33.

$$w_i := w_i - \alpha\frac{\partial J}{w_i} \tag{2.33}$$

À medida que o treinamento progride, se o modelo não estiver subajustado aos dados, a tendência é a convergência da perda para um valor próximo de zero. Com isso, o valor utilizado para atualização dos pesos é cada vez menor e, caso não haja sobreajuste, diz-se que o modelo é capaz de generalizar a partir de novos exemplos.

2.2.8 Teorema Geral da Aproximação

Redes neurais são utilizadas para o aprendizado de funções não-lineares complexas, nas quais não há uma relação explícita entre as variáveis de entrada e a saída desejada. Um modelo linear é capaz de aproximar funções lineares. A baixa complexidade de treinamento e regiões de custo convexas são vantagens de modelos lineares, mas isto restringe a sua aplicação ao aprendizado de funções menos complexas. O teorema da aproximação universal garante que uma rede MLP com uma função de ativação linear em sua saída é capaz de aproximar qualquer função mensurável, de um espaço com dimensões finitas para outro, com um erro não nulo, contanto que sejam fornecidas unidades escondidas suficientes [26], [27] para tal.

A implicação prática deste teorema é que qualquer função pode ser representada por uma MLP. Isto não significa que qualquer processo de treinamento será capaz de calcular os parâmetros correspondentes à função desejada, seja por um sobreajuste aos dados de treinamento e uma baixa capacidade de generalização, seja pela quantidade de dados insuficiente. Redes do tipo *Feedforward*, portanto, consistem em uma ferramenta universal para representação de funções, de modo que sempre existirá uma arquitetura capaz de aproximar uma função desejada.

Não há um procedimento padrão para encontrar uma arquitetura capaz de sintetizar uma determinada função de forma satisfatória. O teorema descrito anteriormente garante que existe uma rede profunda o suficiente e que atinge qualquer patamar de acurácia desejado, mas o processo para alcançar esta arquitetura é completamente empírico. Em suma, uma rede MLP com uma única camada é capaz de representar qualquer função, embora um número exponencial de unidades escondidas seja requerido à medida em que a complexidade das funções aumenta, tornando a camada cada vez maior e com uma capacidade de generalização cada vez menor [28].

Capítulo 3

Ensaios em Malha Fechada

Neste capítulo, descreve-se a modelagem matemática do processo a ser controlado, os experimentos realizados para a coleta dos dados do controlador em malha fechada e a proposição do método de controle utilizando redes neurais.

3.1 A planta

A planta consiste em um aeropêndulo, ilustrado na Figura 3.1. A construção e desenvolvimento são descritos em [15]. O sistema possui dois graus de liberdade e a movimentação do braço se dá por meio de um motor de corrente contínua acoplado a uma hélice. Esta produz uma força em oposição à aceleração gravitacional, o que gera um movimento de rotação no eixo de fixação. O motor é acionado por um *drive* de potência individual e controlado por sinais em modulação por largura de pulso (PWM), conforme descrito em [29].

Figura 3.1: Aeropêndulo utilizado.

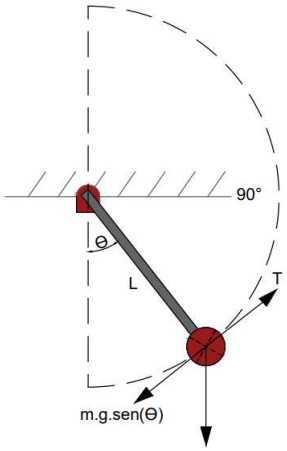
Fonte: BARROS *et al* [15].

Acoplado ao eixo de rotação há um potenciômetro, responsável pela medição do valor do estado a ser controlado, θ, o ângulo em relação ao eixo vertical. A aquisição dos dados, a execução do algoritmo de controle e a geração dos sinais para o atuador são realizadas em um *Arduino Uno R*3.

3.2 Modelagem do Processo

Conforme descrito em [30], adotou-se o método de modelagem baseado em Newton-Euler. A área de atuação do sistema foi limitada, de tal forma que, por decisão de projeto, a hélice aplica forças no pêndulo apenas no sentido anti-horário. O diagrama de forças associado e a área de operação podem ser observados na Figura 3.2.

Figura 3.2: Diagrama de forças atuantes.



Fonte: autoria própria.

Utilizou-se para modelagem matemática do sistema a versão rotacional da segunda lei de Newton. Ademais, considerou-se a força de amortecimento viscoso do ar, atuante de forma contrária ao movimento. A força é, portanto, dependente de $\dot{\theta}$, a velocidade angular, como pode-se observar em Equação 3.1.

$$J.\ddot{\theta} + c.\dot{\theta} + m.g.L.cos(\theta) = T \tag{3.1}$$

Os parâmetros são definidos como:

- J - Momento de Inércia do pêndulo.
- c - Coeficiente de amortecimento viscoso do ar.
- m - Massa do pêndulo.
- g - Aceleração da gravidade.
- L - Comprimento do pêndulo.
- θ - Ângulo do pêndulo em relação ao eixo vertical.
- T - O torque gerado pela rotação da hélice.

Assume-se que o torque em um motor DC pode ser expresso conforme Equação 3.2.

$$T = K_m.L.V^2 \tag{3.2}$$

Onde:

- V - Tensão aplicada no motor.
- K_m - Ganho Torque por tensão do conjunto motor e hélice.

Ao substituir Equação 3.2 na Equação 3.1, obtem-se a Equação 3.3.

$$\ddot{\theta} = \frac{K_m.L}{J}.V^2 - \frac{c}{J}.\dot{\theta} - \frac{m.g.L}{J}.sen(\theta) \tag{3.3}$$

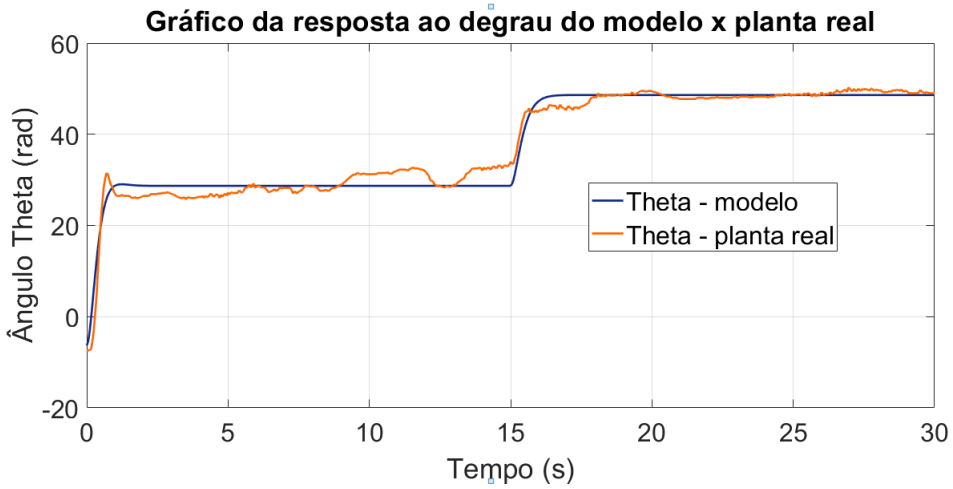
Os comprimentos e pesos foram aferidos com régua e balança. Os demais parâmetros c, J e K_m foram obtidos mediante o uso de uma técnica de otimização de resposta ao degrau, utilizando o algoritmo PSO (*Particle Swarm Optimization*), ou otimização por enxame de partículas. Nela, o problema de ajuste de parâmetros foi otimizado de forma iterativa, por meio da minimização da integral do quadrado da diferença entre a resposta ao degrau do sistema real e a resposta ao degrau do sistema simulado [31]. Os parâmetros encontrados foram sumarizados na Tabela 3.1.

Tabela 3.1: Parâmetros da planta

Parâmetros		
J	0.0114 Kg/m^2	Momento de Inércia em θ
m	0.014 Kg	Massa do corpo do pêndulo
L	0.18 m	Comprimento do Pêndulo θ
g	9.81 m/s^2	Gravidade
K_m	0.6278 N/V	Constante de ganho do motor
c	0.046 Nms/rad	Atrito de viscosidade do ar

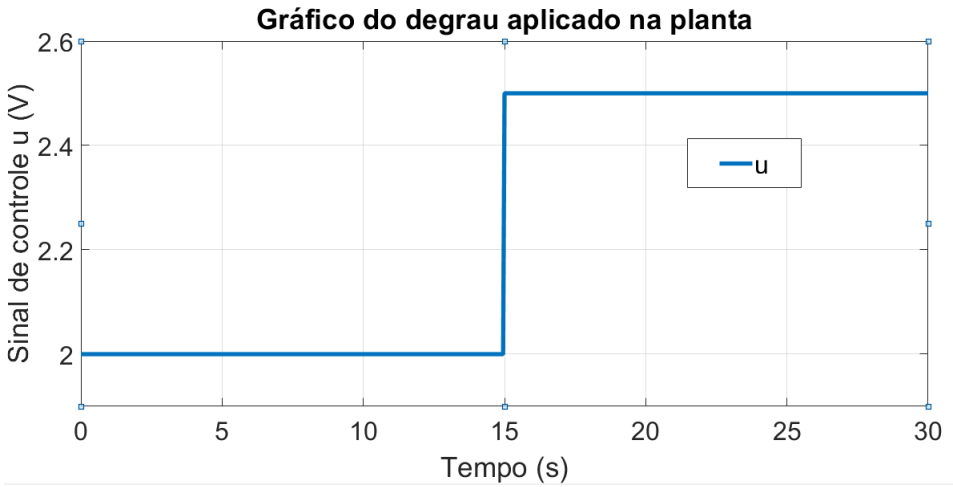
Para validar o modelo, realizou-se a comparação entre a resposta ao degrau simulada e a resposta ao degrau do sistema real, conforme apresentado na Figura 3.3, considerando o sinal de controle aplicado, exibido na Figura 3.4. Observa-se que o modelo e o sistema real apresentam respostas semelhantes, o que sugere a viabilidade de utilização do modelo proposto para simulação da planta.

Figura 3.3: Resposta temporal comparativa após aplicação de degrau de tensão.



Fonte: autoria própria.

Figura 3.4: Degrau de tensão aplicado - sinal de controle.



Fonte: autoria própria.

3.3 Especificações do Controlador

Para projeto do controlador, utilizou-se a *toolbox* NMPC do MATLAB [32]. Por meio dela, é possível simular o controle, em malha fechada, de plantas não lineares sob custos e restrições não lineares. Além disso, é possível planejar trajetórias ótimas ao resolver um problema de otimização não linear com restrições.

Por padrão, os controladores NMPC desta *toolbox* resolvem um problema de programação não linear utilizando a função $fmincon$, que encontra o valor mínimo de uma função associado a restrições conjuntamente com o *solver* SQP (*Sequential Quadratic Programming*).

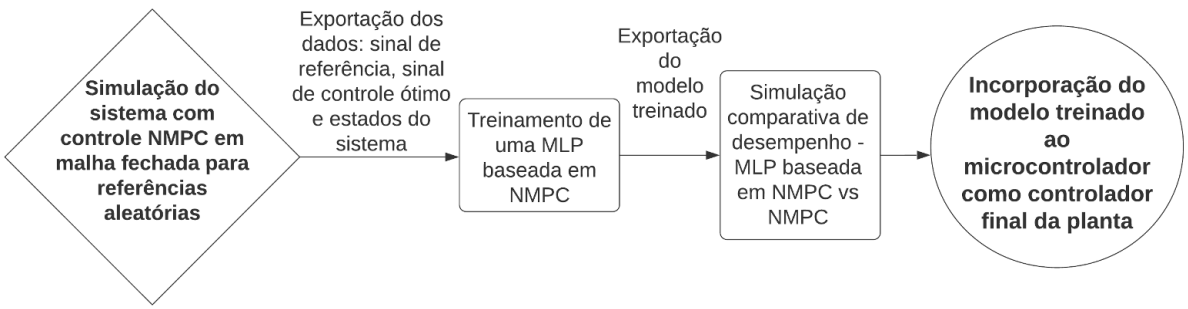
As especificações para o controlador NMPC foram:

- **N - Horizonte de predição** = 20
- **M - Horizonte de controle** = 3
- **Tempo de amostragem** = 0.1s
- u_{min} **- limite inferior do sinal de controle** = 0V
- u_{min} **- limite superior do sinal de controle** = 5V

3.4 Método Proposto

A abordagem proposta neste trabalho pode ser observada na Figura 3.5. Inicialmente, o sistema será simulado a partir do modelo matemático descrito anteriormente para um conjunto de sinais de referência gerados de forma aleatória, de acordo com as limitações operacionais da planta. Os dados relativos ao experimento serão exportados para um ambiente de treinamento de um algoritmo de aprendizagem de máquina, utilizando 80% do conjunto de dados para treino e 20% para teste. O rótulo utilizado para o processo de aprendizagem supervisionada será o sinal de controle e as entradas serão as referências impostas e os estados medidos do sistema.

Figura 3.5: Abordagem de aquisição de dados proposta.



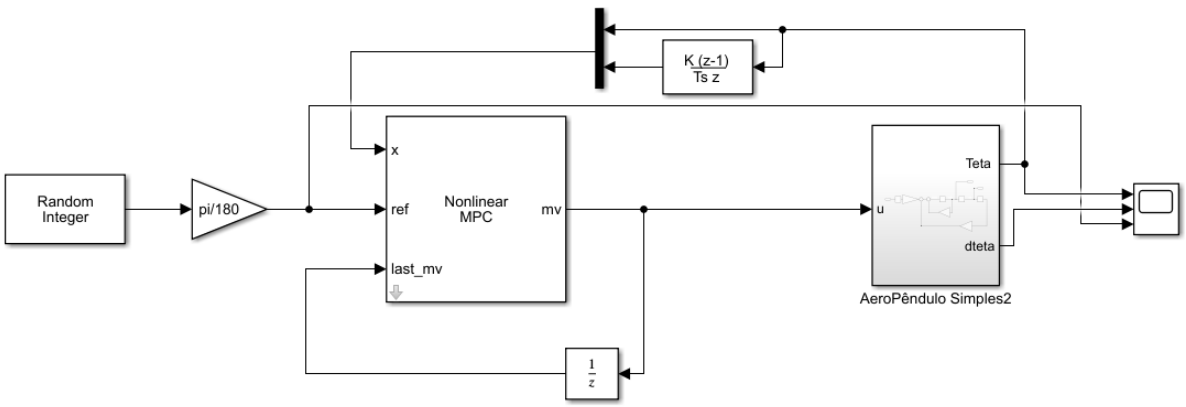
Fonte: autoria própria.

O modelo treinado será exportado para o ambiente de simulação, onde será utilizado para realizar inferências sobre um conjunto de dados não utilizado no treinamento. O desempenho em malha fechada do sistema será comparado com o controlador NMPC tradicional. Ao final do processo, o modelo treinado será incorporado à malha fechada do sistema real, de modo a observar o desempenho do do modelo implementado em *software* no microcontrolador em conjunto com a planta.

3.5 Aquisição dos dados

Diante da impossibilidade de executar o algoritmo NMPC diretamente na planta real, a partir do modelo obtido para o processo e das definições do controlador NMPC na *toolbox* do MATLAB, elaborou-se o diagrama em malha fechada no *software* MATLAB/Simulink para a planta, que pode ser visualizado na Figura 3.6. Como pode-se observar, o valor de $\dot{\theta}$, a velocidade angular, proveniente da saída do bloco que simula o aeropêndulo, não é utilizado. Isso se dá porque no sistema real não há como medir a velocidade angular, havendo a necessidade de estimá-la por meio de uma derivada discreta do estado θ. Este procedimento alternativo para estimação do estado $\dot{\theta}$ também foi incorporado à simulação.

Figura 3.6: Diagrama de controle em malha fechada da planta.



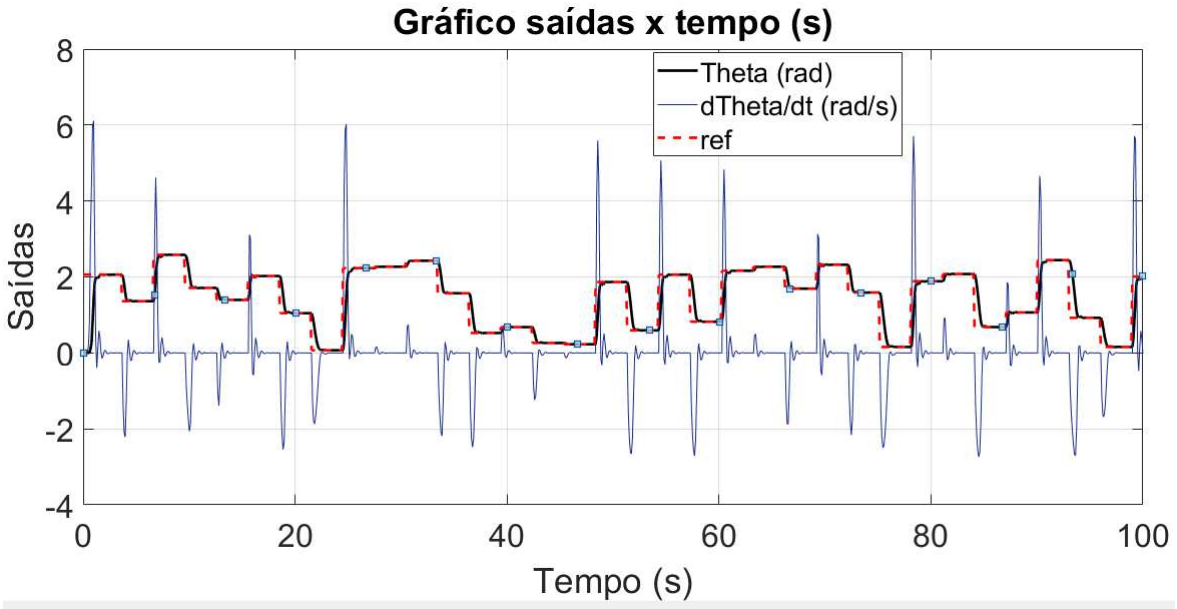
Fonte: autoria própria.

Foram geradas referências aleatórias entre 0 e 149° para o sistema, convertidas para

radianos, com um intervalo três segundos entre as variações. Este intervalo de tempo foi escolhido levando-se em consideração o conhecimento prévio da dinâmica da planta. Para referências variando de forma aleatória com esta frequência de mudança, a saída da planta estabiliza-se no patamar imposto como referência. O experimento teve duração de cem segundos.

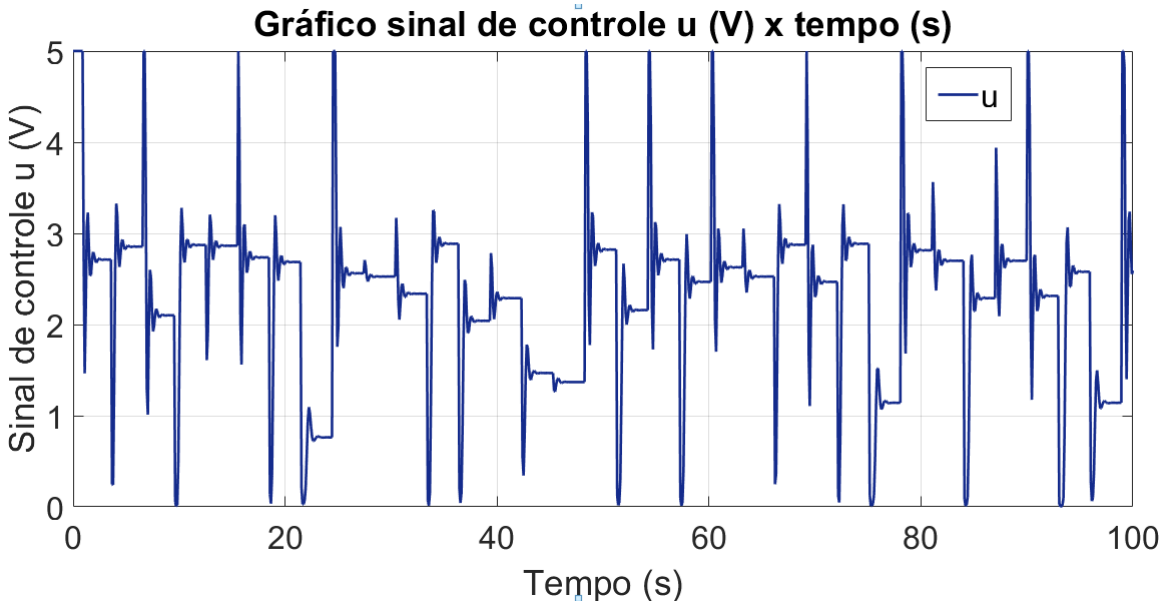
A partir desta simulação, os dados do experimento foram exportados para o ambiente de trabalho do MATLAB. Tendo os estados θ e $\dot{\theta}$, o sinal de controle u e a referência imposta θ_{ref} para cada um dos instantes de amostragem, os dados foram exportados no formato .csv a fim de serem utilizados posteriormente como entrada para o algoritmo de aprendizagem de máquina. A visualização dos dados gerados a partir do controlador NMPC, sinal de controle, estados e referência, podem ser visualizados na Figura 3.7 e na Figura 3.8.

Figura 3.7: Estados e referência relativos à simulação do controlador NMPC.



Fonte: autoria própria.

Figura 3.8: Sinal de controle relativo à simulação do controlador NMPC.



Fonte: autoria própria.

Parte do conjunto de dados, no formato de arquivos .csv, que foi exportada para o ambiente de treinamento de aprendizagem de máquina, pode ser visualizada na Figura 3.9.

Figura 3.9: Conjunto de dados gerado para o sistema em malha fechada.

	teta	dteta	ref	u
0	0.000000e+00	0.000000	2.0595	5.0000
1	3.063700e-10	0.000201	2.0595	5.0000
2	1.102900e-08	0.001206	2.0595	5.0000
3	2.943600e-07	0.006227	2.0595	5.0000
4	7.447100e-06	0.031293	2.0595	5.0000
...	...	...	...	...
1003	1.932700e+00	-0.479330	2.0071	3.1411
1004	1.920300e+00	0.151460	2.0071	3.2387
1005	1.959100e+00	0.577280	2.0071	2.8435
1006	2.004400e+00	0.361090	2.0071	2.5525
1007	2.019200e+00	-0.014923	2.0071	2.5921

1008 rows × 4 columns

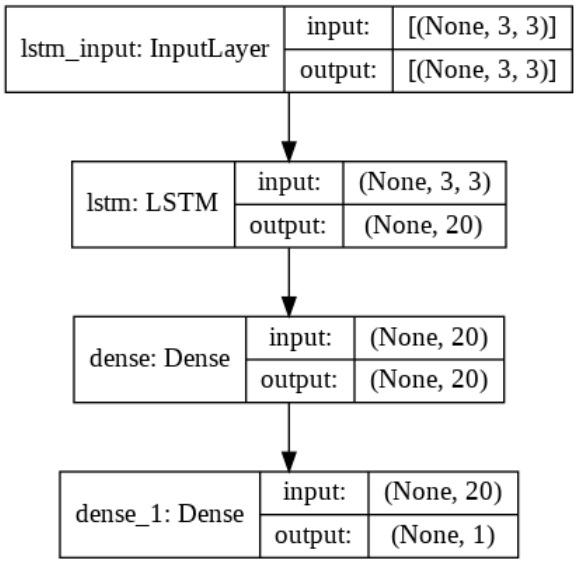
Fonte: autoria própria.

3.6 LSTM baseada em NMPC

Após a aquisição dos dados, definiu-se um modelo LSTM utilizando o Keras [33] como API. O tipo de arquitetura utilizada foi a *many-to-one*, ou seja, várias entradas para composição de uma saída, levando em consideração três amostras $(t-3,t-2,t-1)$ das variáveis de entrada para predição do sinal de controle no instante t.

Após a adequação do conjunto de dados à forma de janela deslizante de tamanho quatro, definiu-se o arquitetura da rede conforme apresentado na Figura 3.10. Esta arquitetura é obtida de forma experimental conforme descreve o teorema da aproximação universal [11]. Partindo de uma arquitetura base, analisa-se parâmetros como subajuste e sobreajuste, de modo a determinar uma arquitetura final capaz de sintetizar a lei de controle desejada a partir dos dados coletados. Na camada que contém unidades LSTM, utilizou-se uma ativação tanh e na camada escondida e de saída, utilizou-se uma ativação linear retificada. Como otimizador, utilizou-se o Nadam com taxa de aprendizagem de 0.001. A função de perda utilizada foi a MSE.

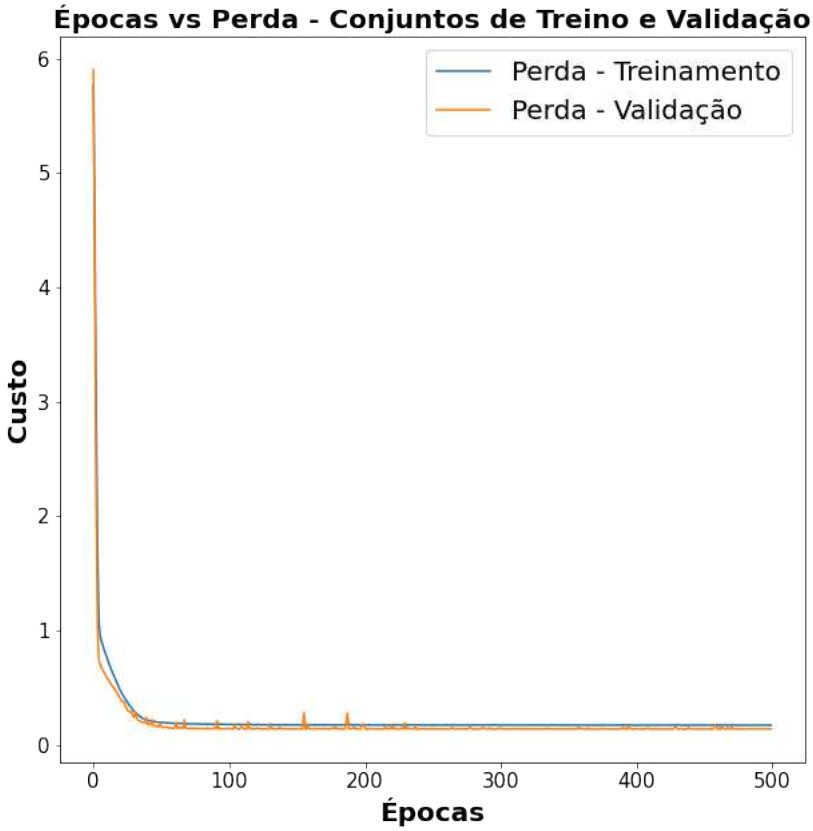
Figura 3.10: Arquitetura da rede utilizando LSTM.



Fonte: autoria própria.

Utilizou-se 80% destes dados para treinamento do modelo, e 20% para validação. O treinamento se deu por 500 épocas e foi realizado no ambiente do Google Colab. O gráfico da perda associada ao modelo pode ser observado na Figura 3.11.

Figura 3.11: Perda associada ao treinamento do modelo LSTM.

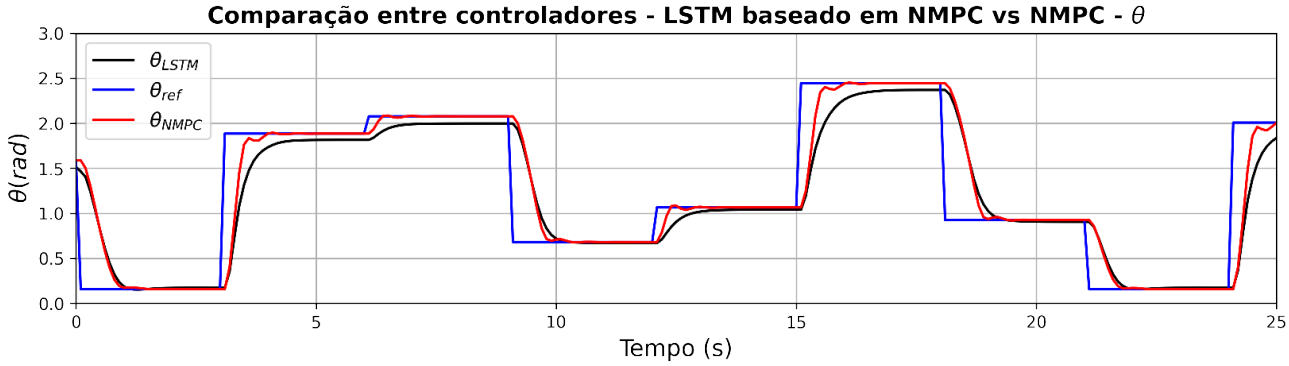


Fonte: autoria própria.

Observa-se, na Figura 3.11 que o modelo tem uma progressão de redução do valor da perda que se estabiliza a partir da época 100, o que elimina a possibilidade de subajuste do modelo. Não há evidências de sobreajuste aos dados de treinamento, já que o valor da perda é muito semelhante para ambos os conjuntos de dados.

O sistema foi simulado em malha fechada, tendo o controlador LSTM baseado em NMPC atuante, utilizando o método de *Runge–Kutta* para a resolução numérica da equação diferencial do modelo. Utilizou-se, então, os dados do conjunto de teste como referência e as condições iniciais dos estados associadas a ele. O modelo pré-treinado foi utilizado para calcular as ações ótimas de controle. O desempenho comparativo entre o LSTM baseado em NMPC e o NMPC tradicional, em malha fechada, pode ser observado na Figura 3.12.

Figura 3.12: Desempenho em malha fechada para o estado θ dos controladores LSTM baseado em NMPC e NMPC.



Fonte: autoria própria.

Observa-se um erro, em regime, para o controlador LSTM baseado em NMPC. Isto pode ser atribuído à necessidade de ajuste de hiperparâmetros da rede neural, evidenciada pela erro médio observado ao longo das últimas épocas de treinamento, conforme Figura 3.11. Existe um erro associado ao cálculo das ações ótimas de controle que conduz à diferença sutil do estado θ em relação à referência imposta.

Os pesos do modelo Keras foram exportados no formato $.h5$ para importação no Simulink. No entanto, por incompatibilidade momentânea com o Keras e o breve tempo disponível para maiores investigações, não foi possível incorporar o modelo LSTM à malha fechada da simulação.

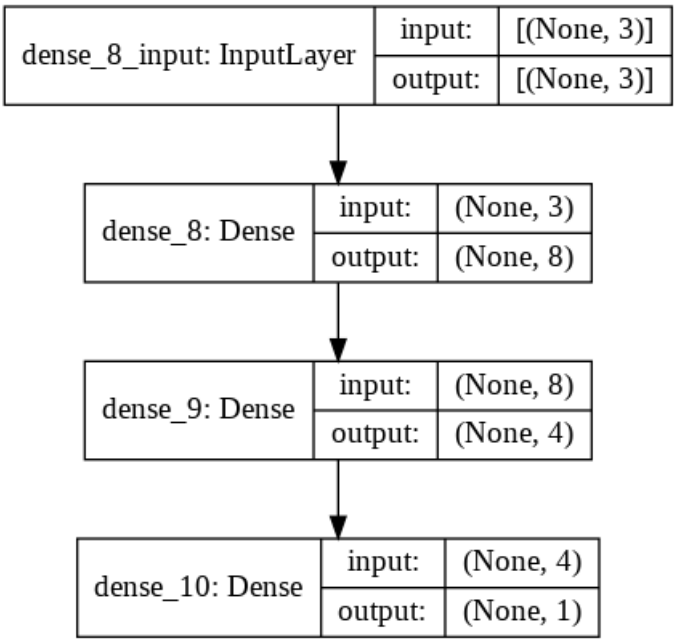
Investigou-se, também, a possibilidade de realizar o *deploy* do modelo no microcontrolador utilizado, o *Arduino Uno R*3, por meio do TensorFlow Lite [34]. O TFLite possibilita a execução de modelos de aprendizagem de máquina em microcontroladores e outros dispositivos. Constatou-se que não há compatibilidade desta API com o *Arduino Uno R*3.

Existe a possibilidade de definir de forma manual o modelo, juntamente com os pesos pré-treinados, no código fonte do microcontrolador e realizar as inferências necessárias a partir deste modelo. Todavia, devido à arquitetura recorrente e sua estrutura complexa entre camadas, não foi possível realizar esta etapa a tempo. Assim, não foi possível avaliar o desempenho do modelo de inferência LSTM baseado em NMPC na planta real, o que conduziu à busca por uma arquitetura alternativa.

3.7 MLP baseada em NMPC

Semelhantemente à abordagem utilizando LSTM, utilizou-se os dados exportados para treinamento de uma rede perceptron multicamadas, MLP. Foram necessárias experimentações para determinar uma arquitetura compacta capaz de sintetizar a lei de controle NMPC e generalizar de forma satisfatória. A arquitetura final pode ser observada na Figura 3.13. Utilizou-se a ativação tanh nas camadas escondidas e ReLU nas camadas de entrada e saída. Não foi necessário aplicar métodos de regularização para evitar sobreajuste. A função de custo utilizada foi a MSE e o otimizador Nadam com taxa de aprendizagem igual a 0.01.

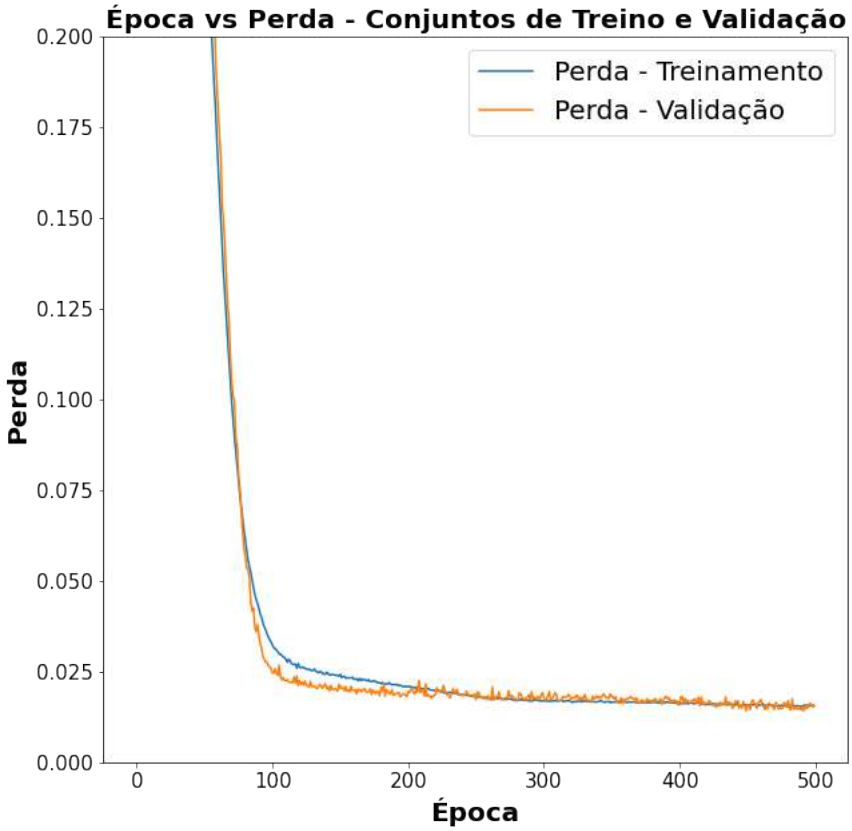
Figura 3.13: Arquitetura da MLP utilizada.



Fonte: autoria própria.

O modelo foi implementado utilizando a mesma API do LSTM, o Keras, e treinado por 500 épocas com um *batch size* de 32. Os resultados em termos da perda revelam que o modelo apresentou uma boa capacidade de generalização, perceptível pelo progresso da perda tanto para os dados de treinamento, quando para os dados de validação. Além disso, a progressão observada na Figura 3.14 indica que o modelo não está subajustado aos dados, visto que a perda decai para um valor muito próximo de zero, e corrige o platô observado no treinamento da rede LSTM.

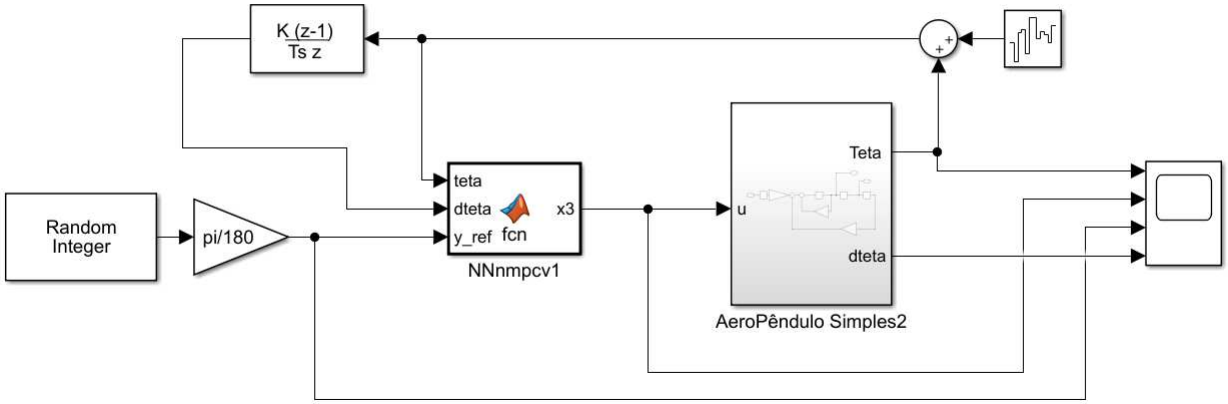
Figura 3.14: Progressão da perda ao longo das épocas de treinamento.



Fonte: autoria própria.

Posteriormente, os pesos do modelo treinado foram exportados para o ambiente do Simulink, sendo utilizados para definir um bloco que recebe como entrada os estados do sistema e a referência e calcula os sinais de controle necessários. O diagrama elaborado no ambiente de simulação pode ser observado na Figura 3.15. O bloco de função $NNnmpcv1$ substitui o controlador original NMPC e nele estão as definições do modelo MLP pré-treinado.

Figura 3.15: Diagrama de simulação em malha fechada utilizando a MLP baseada em NMPC.



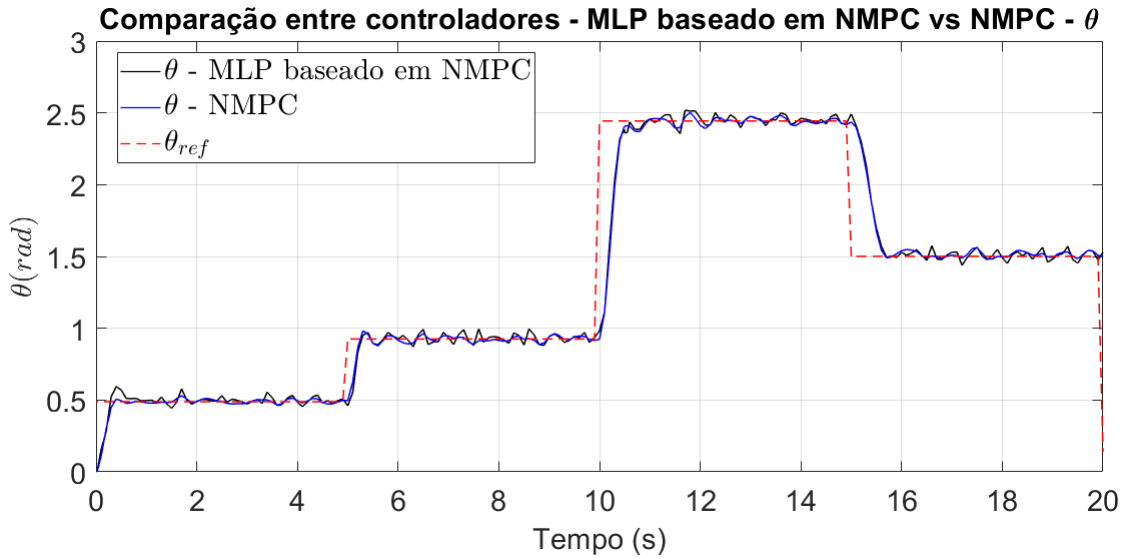
Fonte: autoria própria.

No diagrama exibido na Figura 3.15, para tornar a simulação mais próxima dos valores medidos na planta real, adicionou-se um ruído branco de potência 0.00005 ao estado θ, que naturalmente também é incorporado ao estado $\dot{\theta}$. Posteriormente, o objetivo é comparar o desempenho deste controlador com o NMPC mediante a aplicação de um degrau de referência ao modelo.

3.8 Análise dos Resultados de Simulação

A comparação da saída θ para os dois controladores é apresentada na Figura 3.16. Como pode-se observar, o controlador pré-treinado acompanha as dinâmicas em malha fechada de forma muito semelhante ao controlador NMPC tradicional para o estado θ.

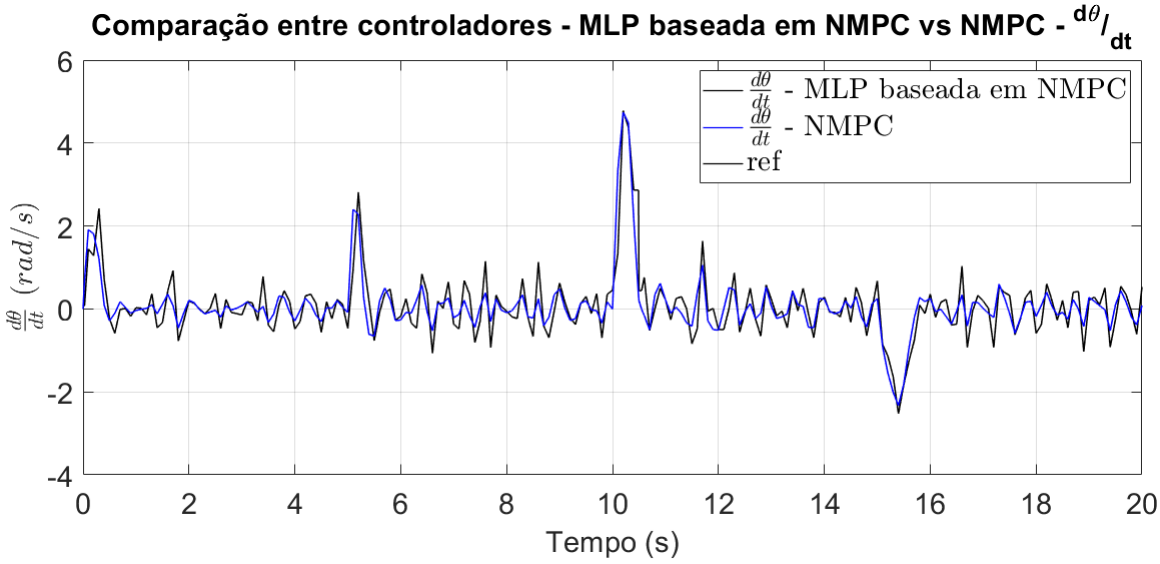
Figura 3.16: Comparação do rastreamento de referência em malha fechada.



Fonte: autoria própria.

Na Figura 3.17 está a comparação entre os estados $\dot{\theta}$ para as duas abordagens. O estado $\dot{\theta}$ medido a partir da atuação do controlador MLP também acompanha as dinâmicas do controlador NMPC para referências aleatórias impostas ao sistema.

Figura 3.17: Comparação em malha fechada para a velocidade angular $\dot{\theta}$.

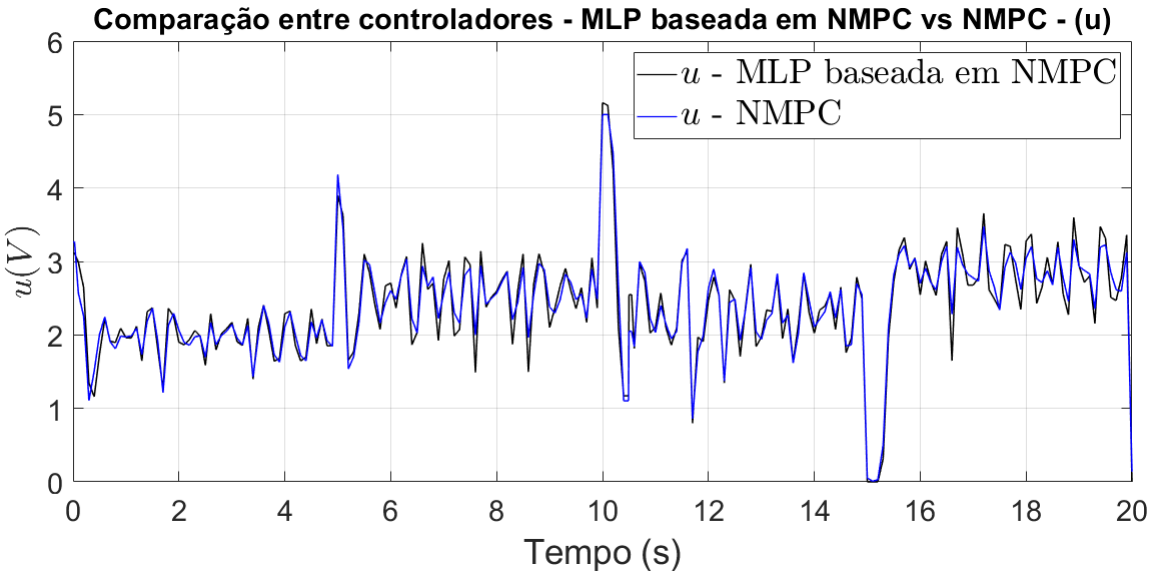


Fonte: autoria própria.

A comparação entre os sinais de controle gerados pelos dois controladores pode ser observada na Figura 3.18. Observa-se uma similaridade entre os sinais de controle calculados,

o que indica, *a priori*, que o controlador sintetizado apresenta comportamento similar ao controlador não-linear original.

Figura 3.18: Comparação entre os sinais de controle - MLP baseada em NMPC vs NMPC.



Fonte: autoria própria.

Visando uma comparação mais efetiva entre os controladores, utilizou-se a métrica IAE (*Integral Absolut Error*)[35], definida em (3.4).

$$IAE = \int_0^{t_{final}} |e| dt \tag{3.4}$$

A variação total, *Total Variation* - TV [36], é definido em (3.5).

$$TV = \sum_{t=0}^{Tfinal} |u(t+1) - u(t)| \tag{3.5}$$

Na Tabela 3.2 estão apresentado os valores numéricos das métricas escolhidas para cada ensaio com controlador apresentado.

Tabela 3.2: Métricas de desempenho.

Controlador	IAE-Theta	TV-u
NMPC	26.662	88.531
MLP baseado em NMPC	26.789	110.473

Analisando os gráficos e as métricas expostas, observa-se que o controlador MLP baseado

em NMPC apresenta um comportamento de saída semelhante ao do controlador NMPC no contexto de simulação. O valor do IAE para os dois controladores foi similar, todavia fica evidente que há uma leve diferença no esforço de controle aplicado, e isto pode ser confirmado observando os valores distintos para o TV observados na Tabela 3.2. O valor superior do TV referente à MLP baseada em NMPC indica que este controlador aplicar um maior esforço de controle para alcançar a mesma estabilização.

A diferença principal entre a MLP baseada em NMPC e o NMPC é a redução do tempo de cálculo do sinal de controle. Diante das limitações de *hardware* que impossibilitam a execução do algoritmo NMPC na planta real, mediu-se o tempo de cálculo da ação de controle pelo NMPC e pela MLP baseada em NMPC no contexto de simulação. Ambos foram executados em um computador com um processador i5-8265 com 4 núcleos e frequência 1.60 GHz e memória RAM de 16 GB. Mil ações de controle foram calculadas em sequência visando a estimativa do tempo médio de processamento. O resultado pode ser observado na Tabela 3.3

Tabela 3.3: Tempo de processamento de cada método.

Parâmetro	NMPC	MLP baseado em NMPC	Melhoria
Tempo Mínimo	40.9 ms	6.90μ s	592718.8 %
Tempo Médio	47.7 ms	10.3μ s	460637.4 %

Observando a Tabela 3.3 é possível constatar uma melhoria considerável, no que tange o tempo de processamento, em simulação, apresentada pelo controlador MLP quando comparado ao controlador NMPC tradicional. A viabilidade da utilização do MLP baseado em NMPC, no lugar do NMPC tradicional, é reforçada diante da similaridade entre as dinâmicas do sistema simulado. Além disso, houve uma redução expressiva no tempo de cálculo para o sinal de controle utilizando o NMPC emulado, conforme observa-se na Tabela 3.3.

3.9 Aplicação do NMPC à planta real

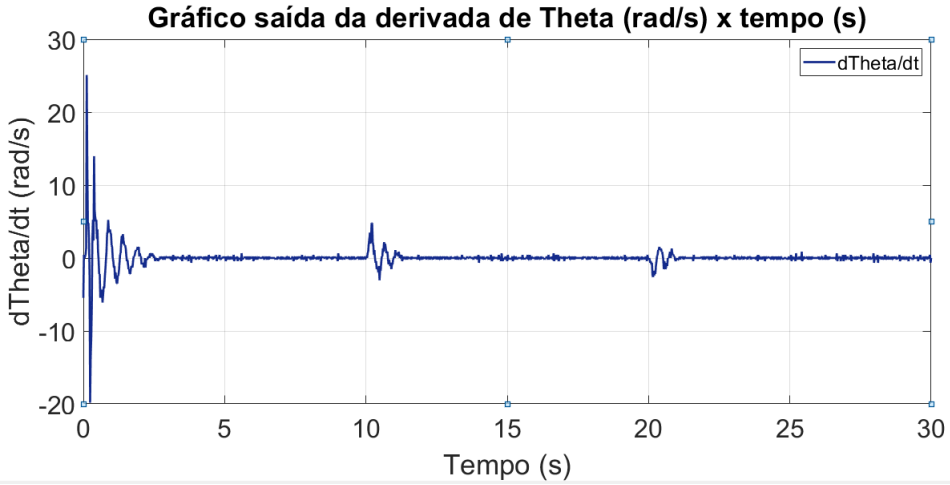
Os pesos matriciais relativos ao controlador MLP baseado em NMPC foram exportados e o modelo foi definido manualmente no microcontrolador. Este modelo foi aplicado à planta real do aeropêndulo, obtendo as saídas apresentadas na Fig. 3.19 e na Fig. 3.20.

O sinal de controle pode ser observado na Fig. 3.21. Devido às limitações de *hardware* do microcontrolador disponível, não foi possível aplicar o NMPC original no sistema real.

Figura 3.19: Resposta temporal em malha fechada para o estado θ - controlador MLP baseado em NMPC.

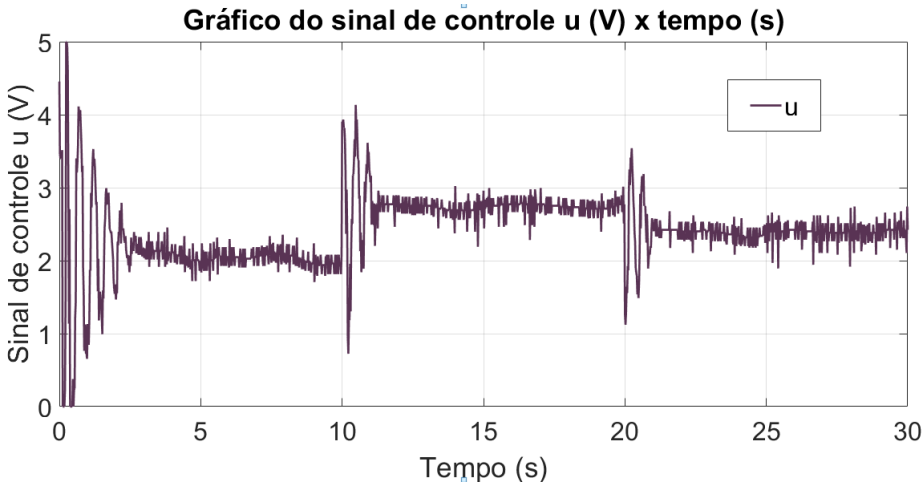
Fonte: autoria própria.

Figura 3.20: Estado $\dot{\theta}$ utilizando o controlador MLP baseado em NMPC em malha fechada.



Fonte: autoria própria.

Figura 3.21: Sinal de controle - MLP baseado em NMPC.



Fonte: autoria própria.

Na Tabela 3.4 estão apresentado os valores numéricos das métricas escolhidas IAE e TV para o experimento na planta real apresentado.

Tabela 3.4: Métricas de desempenho.

Controlador	IAE-Theta	TV-u
MLP baseado em NMPC	22.219	239.887

Analisando as Figuras 3.19, 3.20 e 3.21, é possível observar que o controlador MLP baseado em NMPC não foi suficiente para controle efetivo e estabilização do sistema do aeropêndulo. Pode-se observar que o controlador aplicado na planta real resultou em uma resposta oscilatória mais agressiva ao verificar a evolução do estado θ, especialmente para o primeiro degrau aplicado ao sistema. Isto é reforçado pelo valor elevado do TV. O sistema apresenta um sobressinal que dura aproximadamente dois segundos. Como os dados utilizados para treinamento da MLP são provenientes da simulação do sistema, as condições de operação reais, como ruídos no momento da medição, podem ser apontados como causadores deste comportamento oscilatório na resposta real. Apesar disso, o valor IAE resultou em valores menores do que os simulados, dada a maior velocidade de estabilização do sistema.

Capítulo 4

Conclusão

A abordagem descrita neste trabalho propõe um procedimento de treinamento de uma rede neural MLP baseada em um controlador NMPC, aplicado a um estudo de caso com um aeropêndulo. Atendendo os objetivos gerais propostos, a principal vantagem da metodologia descrita é a redução do tempo de cálculo do sinal de controle a cada instante de amostragem, mediante a emulação do controlador original por uma rede neural artificial.

Este desenvolvimento foi motivado pelo desafio de controle de um aeropêndulo, sistema tipicamente não linear. O algoritmo de controle proposto mostrou-se capaz de operar conforme os requisitos estabelecidos, sendo computacionalmente simples a ponto de poder ser implementado em um microcontrolador de 8bits/16MHz.

Ao cumprir os objetivos específicos propostos, o estudo de caso do aeropêndulo indicou que o algoritmo proposto proporcionou uma diminuição de até 6000 vezes no tempo de cálculo das ações de controle em comparação com uma estratégia clássica de NMPC. Ademais, observou-se uma similaridade nas dinâmicas em malha fechada entre o controlador NMPC original e o MLP baseado em NMPC, obtendo sucesso na estabilização do sistema para quaisquer referências impostas no contexto de simulação. Na aplicação à planta real, o desempenho em malha fechada não foi o mesmo, e isto pode ser explicado devido à impossibilidade de aplicação do controlador NMPC à planta real para uma geração de dados mais fidedigna à realidade.

O método descrito foi aplicado a um sistema SISO. No entanto, pode ser aplicado a sistemas MIMO não lineares, especialmente quando existe uma limitação na capacidade

computacional. No caso de sistemas MIMO, basta ajustar a arquitetura da rede para realizar uma regressão multivariável.

O trabalho teve como fruto um artigo aceito para publicação no INDUSCON (*IEEE/IAS International Conference on Industry Applications 2021*). Isto endossa a relevância do trabalho e da aplicação de métodos que exploram o elo entre aprendizagem de máquina e técnicas de controle.

Para trabalhos futuros, sugere-se a utilização de sistemas MIMO com linearidades mais severas, como descontinuidades. Também recomenda-se a realização de testes com micro-controladores com suporte ao TensorFlow Lite, visto que este permite o *deploy* de modelos Keras mais complexos, sem a necessidade da importação manual dos pesos e definição da arquitetura da rede.

Referências bibliográficas

1 WANG, Liuping. *Model Predictive Control System Design and Implementation Using MATLAB*. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2009. 2-4 p.

2 QIN, Joe; BADGWELL, Thomas. An overview of industrial model predictive control technology. *AIChE Symposium Series*, v. 93, 01 1997.

3 BEMPORAD, Alberto; MORARI, Manfred. Robust model predictive control: A survey. *Robustness in identification and control*, Springer, p. 207–226.

4 ISIDORI, Alberto. *Nonlinear Control Systems*. [S.l.]: Springer London, 1995.

5 MARINO, Riccardo; TOMEI, Patrizio. *Nonlinear Control Design: Geometric, Adaptive and Robust*. GBR: Prentice Hall International (UK) Ltd., 1996. ISBN 0133426351.

6 NIJMEIJER, Henk; SCHAFT, Arjan van der. Nonlinear dynamical control systems. Springer New York, 1990.

7 GROS, Sébastien; ZANON, Mario; QUIRYNEN, Rien; BEMPORAD, Alberto; DIEHL, Moritz. From linear to nonlinear MPC: bridging the gap via the real-time iteration. *International Journal of Control*, Informa UK Limited, v. 93, n. 1, p. 62–80, set. 2016.

8 RAHA, Arnab; CHAKRABARTY, Ankush; RAGHUNATHAN, Vijay; BUZZARD, Gregery T. Embedding approximate nonlinear model predictive control at ultrahigh speed and extremely low power. *IEEE Transactions on Control Systems Technology*, Institute of Electrical and Electronics Engineers (IEEE), v. 28, n. 3, p. 1092–1099, maio 2020.

9 MAHMOUD, Magdi S.; SABIH, Muhammad; ELSHAFEI, Moustafa. Using OPC technology to support the study of advanced process control. *ISA Transactions*, Elsevier BV, v. 55, p. 155–167, mar. 2015.

10 GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning*. [S.l.]: MIT Press, 2016.

11 HORNIK, Kurt. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, Elsevier BV, v. 4, n. 2, p. 251–257, 1991.

12 WANG, Songda; DRAGICEVIC, Tomislav; GONTIJO, Gustavo Figueiredo; CHAUDHARY, Sanjay K.; TEODORESCU, Remus. Machine learning emulation of model predictive control for modular multilevel converters. *IEEE Transactions on Industrial Electronics*, p. 1–1, 2020.

13 COTORRUELO, Andres; RAMIREZ, Daniel R.; LIMON, Daniel; GARONE, Emanuele. Nonlinear mpc for tracking for a class of non-convex admissible output sets. *IEEE Transactions on Automatic Control*, p. 1–1, 2020.

14 KUMAR, Steven Spielberg Pon; TULSYAN, Aditya; GOPALUNI, Bhushan; LOEWEN, Philip. A deep learning architecture for predictive control. *IFAC-PapersOnLine*, Elsevier BV, v. 51, n. 18, p. 512–517, 2018.

15 BARROS, A. F. Jr; LIMA, R. B. C.; Galdino, R. S.; Maciel, V. R. B.; Dias, C. C. *Construção e Controle de um Aero Pêndulo, Uma Abordagem Baseada em Modelo.* [S.l.: s.n.], 2019.

16 FAROOQ, Umar; GU, Jason; EL-HAWARY, Mohamed E.; LUO, Jun; ASAD, Muhammad Usman. Observer based fuzzy lmi regulator for stabilization and tracking control of an aeropendulum. In: *2015 IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE).* [S.l.: s.n.], 2015. p. 1508–1513.

17 FINDEISEN, Rolf. *Nonlinear Model Predictive Control: A Samples-Data Feedback Perspective*. 2-5 p. Tese (Doutorado), 12 2004.

18 KOLLURI, Suryanarayana; ADURU, Sai; PATHAK, Manan; BRAATZ, Richard; SUBRAMANIAN, Venkat. Real-time nonlinear model predictive control (nmpc) strategies using physics-based models for advanced lithium-ion battery management system (bms). *Journal of The Electrochemical Society*, v. 167, p. 063505, 04 2020.

19 MITCHELL, Tom M. *Machine Learning*. New York: McGraw-Hill, 1997. 1-3 p.

20 BRANCO, Henrique; XIMENES, Lucas; OLIVEIRA, Roberto. *Overfitting e underfitting em Machine Learning - ABRACD - ASSOCIAÇÃO BRASILEIRA DE CIÊNCIA DE DADOS*. 2020. Disponível em: <https://abracd.org/overfitting-e-underfitting-em-machine-learning/>.

21 FIORIN, Daniel V.; MARTINS, Fernando R.; SCHUCH, Nelson J.; PEREIRA, Enio B. Aplicações de redes neurais e previsões de disponibilidade de recursos energéticos solares. *Revista Brasileira de Ensino de Física*, FapUNIFESP (SciELO), v. 33, n. 1, p. 01–20, mar. 2011.

22 OLGAC, A; KARLIK, Bekir. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence And Expert Systems*, v. 1, p. 111–122, 02 2011.

23 HOCHREITER, Sepp; SCHMIDHUBER, Jürgen. Long short-term memory. *Neural Computation*, v. 9, n. 8, p. 1735–1780, 1997.

24 UNDERSTANDING LSTM Networks. Disponível em: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>.

25 SYDORENKO, Iryna. *What Is a Dataset in Machine Learning*. Label Your Data, 2021. Disponível em: <https://labelyourdata.com/articles/what-is-dataset-in-machine-learning/?utm_source=blog&utm_medium=facebook&utm_campaign=dataset>.

26 HORNIK, Kurt; STINCHCOMBE, Maxwell; WHITE, Halbert. Multilayer feedforward networks are universal approximators. *Neural Networks*, Elsevier BV, v. 2, n. 5, p. 359–366, jan. 1989.

27 CYBENKO, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, Springer London, v. 2, n. 4, p. 303–314, dez. 1989.

28 BARRON, Andrew. Barron, a.e.: Universal approximation bounds for superpositions of a sigmoidal function. ieee trans. on information theory 39, 930-945. *Information Theory, IEEE Transactions on*, v. 39, p. 930 – 945, 06 1993.

29 OLIVEIRA, Arthur D. B.; BARROS, Stayner N.; LIMA, Rafael B. Correia. Controlador nmpc embarcado para aeropêndulo utilizando mlp. *IEEE/IAS - International Conference on Industry Applications*, 08 2021 (no prelo).

30 BARROS, Stayner N.; LIMA, Rafael B. C. Controlador PID ótimo baseado em PSO aplicado a um aeropêndulo. In: *Anais do Congresso Brasileiro de Automática 2020*. [S.l.]:

31 Meiying, Y.; Xiaodong, W. Pso-based parameter estimation of nonlinear systems. In: *2007 Chinese Control Conference*. [S.l.: s.n.], 2007. p. 533–536.

32 NMPC Toolbox. Disponível em: <https://www.mathworks.com/help/mpc/ug/nonlinear-mpc.html>.

33 TEAM, Keras. *Keras documentation: Keras API reference*. Disponível em: <https://keras.io/api/>.

34 GUIA do TensorFlow Lite. Disponível em: <https://www.tensorflow.org/lite/guide?hl=pt-br>.

35 Akhtar, M. A.; Saha, S. Iae and ise performance criterion based loop filter tuning of transport delay-phase locked loop (td-pll) for single phase grid connected inverters. In: *2018 8th IEEE India International Conference on Power Electronics (IICPE)*. [S.l.: s.n.], 2018. p. 1–5.

36 SKOGESTAD, Sigurd. Simple analytic rules for model reduction and pid controller tuning. *Journal of Process Control*, v. 13, n. 4, p. 291 – 309, 2003.