UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

ENGENHARIA FÍSICA

Geração de Distribuições Cinemáticas de Mésons π e K no Processo $D^0 \to \pi K$ em Colisões PbPb do LHC Utilizando Autoencoders Variacionais

Davi Leon de Souza

Orientador: César Augusto Bernardes (IF - UFRGS)
Coorientador: Breno Orzari (Unesp)

Sumário

Lista de Figuras

1. INTRODUÇÃO

A física de partículas é um campo intrincado e fascinante, que busca desvendar os segredos do universo ao explorar os constituintes fundamentais da matéria e as forças que os governam. Nesse cenário complexo, os mésons emergem como partículas de interesse. Por exemplo, essas partículas podem ser produzidas em quantidades significativas em colisões ultrarrelativísticas de íons pesados, desempenhando um papel vital na investigação de fenômenos subatômicos e na validação das interações da física de partículas. Identificar e estudar essas partículas de maneira eficaz representa um desafio significativo, exigindo abordagens inovadoras e técnicas avançadas de análise de dados.

As colisões de íons pesados, como as realizadas no Grande Colisor de Hádrons (LHC) [1], oferecem um ambiente propício para a criação de mésons. Elas se destacam por serem capazes de reproduzir caraterísticas parecidas com as do início do universo, alguns microssegundos após o Big Bang. De maneira simplificada, o processo ocorre da seguinte maneira (ver Fig. 1.1): dois núcleos pesados interagem, provenientes de dois feixes de núcleos pesados em um acelerador de partículas. Cada par de núcleons (prótons e nêutrons) possui uma energia no centro de massa da ordem de trilhões de elétron-volts, no local da interação, caracterizada como uma “bola de fogo”. Esse ambiente de densidade de energia da ordem de 1 $\mathrm{GeV/fm^3}$ faz com que os quarks e glúons tenham a magnitude da interação enfraquecida, desprendendo-se dos prótons e nêutrons e criando uma substância que possui propriedades mais próximas das de um fluido, conhecida como plasma de quarks e glúons (QGP). Após alguns instantes (ordem de $10\,\mathrm{fm}/c$, sendo c a velocidade da luz no vácuo), essa região expande devido à grande pressão, diminuindo a densidade de energia e fazendo com que essas partículas se recombinem em partículas compostas.

Os hádrons representam alguns dos resultados dessa recombinação e têm

características distintas entre si. Alguns se deslocam por grandes distâncias (até alguns metros) e podem ser facilmente recebidos pelos detectores, que tem a sua primeira camada de detecção a 3 cm da colisão. Em contrapartida, outros hádrons decaem muito rapidamente, impossibilitando a sua detecção. Nesses casos, a metodologia passa a ser detectar as partículas filhas de seu decaimento e usar as suas características cinemáticas para identificar esse hádron de curto tempo de vida. É nesse caso que a partícula de interesse deste trabalho, o méson D^0, se enquadra, percorrendo da ordem de 100 $\mu\mathrm{m}$ antes de decair em outras partículas.

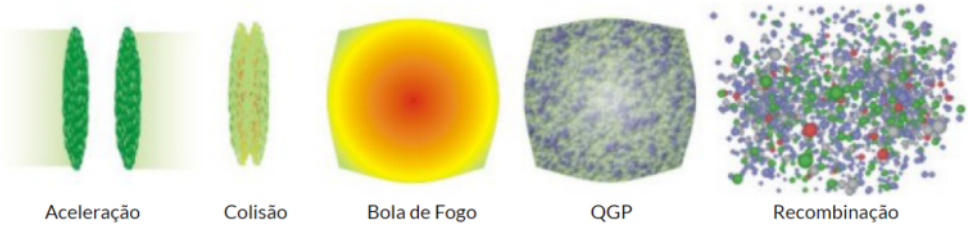

Figura 1.1: Etapas resultantes de uma colisão de íons pesados (fonte: [2]).

Após a colisão e a detecção das partículas, os dados cinemáticos das partículas que deixaram sinais nos detectores são processados via algoritmos computacionais para determinar quais foram os produtos da colisão.

Além desse processo realizado em laboratório, existem algoritmos utilizados para simular colisões artificialmente, buscando representar, da maneira mais fiel possível, colisões reais. O mais comumente escolhido é a simulação de Monte Carlo, que emprega distribuições de probabilidade para simular as colisões, representando trajetórias, interações e detecções de partículas nesse processo. Apesar de ser um método flexível e capaz de simular cenários complexos como este, o Monte Carlo possui limitações, dentre as quais se destacam o elevado tempo de processamento e o alto ruído (diferença com relação aos dados reais) do output. Isso ocorre porque a quantidade de eventos que precisam ser simulados é muito grande (evidência disso é o período das colisões, que é da ordem de dezenas de nanossegundos), fazendo com que o algoritmo de Monte Carlo possa levar até 2 meses para concluir a simulação. É nesse contexto que este trabalho se concentra, propondo um novo método para esse processo: reconstruir o méson D^0 a partir de seu decaimento em um píon e um káon carregados utilizando autoencoders variacionais, substituindo a simulação de Monte

Carlo nesse contexto.

o período das colisões que devem ser simuladas é da ordem de dezenas de nanossegundos

Os Autoencoders Variacionais (VAEs) são modelos de machine learning que se destacam pela capacidade de aprender características profundas nos dados. Eles pertencem à classe de redes neurais artificiais generativas (capazes de gerar novos dados no output) e têm a capacidade de capturar e comprimir informações cruciais contidas nos dados de entrada, permitindo a criação de representações mais compactas, contendo apenas as características mais valiosas da entrada. No contexto deste projeto, os VAEs serão treinados para gerar o momento transversal das partículas provenientes de um dos modos de decaimentos dos mésons D^0, isto é, em um píon (π) e um káon (K). Esse será o primeiro passo para a reconstrução completa do méson D^0 usando este método. Esse processo será aprofundado na Metodologia (capítulo 3). Os VAEs representam um bom candidato para essa missão, visto que já foram usados para aplicações similares no campo da física de partículas e obtiveram um bom resultado. Um exemplo disso é um trabalho em que autoencoders variacionais convolucionais foram utilizados para gerar jatos hadrônicos [3]. Para entender mais sobre as demais aplicações dessa rede neural, a Ref. [4] pode ser acessada.

Comparado ao Monte Carlo, espera-se obter ganho de eficiência em tempo de execução e diminuição de ruído, uma vez que o VAE será treinado para gerar apenas as grandezas de interesse para a recriação do méson D^0, enquanto a arquitetura da simulação de Monte Carlo emula o evento da colisão como um todo, focado em todos os seus subprodutos, e não apenas no méson D^0.

Conforme será descrito na seção de Metodologia (capítulo 3), os dados utilizados neste projeto são advindos das simulações de Monte Carlo, porém, sendo comprovada a efetividade do VAE em reconstruir o momento transversal das partículas filhas do méson D^0 (e, posteriormente, de reconstruí-lo também) nessa amostra, no futuro (em outro projeto), esse método deverá ser aplicado em dados reais, substituindo a simulação de Monte Carlo. Em outras palavras, o objetivo do projeto é validar uma prova de conceito em dados simulados que poderá, posteriormente, substituir essas mesmas simulações e levar ganhos às análises em dados reais. Essa abordagem não apenas oferecerá insights mais profundos sobre a natureza dos mésons D^0, mas também contribuirá

para o avanço da pesquisa em física de partículas, fornecendo uma ferramenta valiosa para futuros experimentos.

2. REFERENCIAL TEÓRICO

2.1. FÍSICA DE PARTÍCULAS E O MÉSON D^0

Após a colisão de dois núcleos de chumbo (Pb), o méson D^0 é concebido pela união de um quark charm (*c*) e um antiquark up (*u*) [5] enquanto o QGP esfria. Após a sua criação, a curta vida desse méson é marcada por um rápido decaimento que, da ordem de 4% dos casos, resulta na produção das partículas filhas π e *K* carregados, como ilustrado na Fig. 2.1. Ele viaja, em média, poucas centenas de micrometros antes de decair, impossibilitando a sua detecção direta, visto que as primeiras camadas de material sensível dos detectores instalados no LHC, o CMS (Compact Muon Solenoid) [6], estão a uma distância de alguns centímetros do ponto de colisão. Por outro lado, os píons e káons percorrem distâncias maiores e podem ser detectados. Ao atingir o detector, essas partículas filhas deixam uma “impressão digital”, que podem ser traduzidas em uma série de características cinemáticas, como velocidades, momentos, ângulos de incidência, etc. Essas características podem então ser analisadas para entender como a colisão ocorreu. Em particular, a grandeza de interesse deste trabalho é o momento transversal dos píons e dos káons, como será descrito na sequência. Portanto, para reconstruírmos as características cinemáticas do méson D^0, podemos fazer uma espécie de engenharia reversa, usando as características cinemáticas das partículas filhas do seu decaimento para recriar as suas componentes.

2.1.1. COMPONENTES DO MÉSON D^0 E DAS SUAS PARTÍCULAS FILHAS

Os mésons D^0, bem como as suas partículas filhas píons e káons, podem ser inteiramente representados pelo seus quadrimomentos P = (momento transversal,

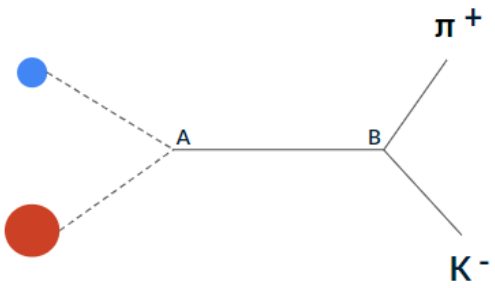

Figura 2.1: Processo de formação e decaimento do D^0. A bola azul representa um antiquark up e a vermelha um quark charm. No ponto A, o méson é formado através da união dos quarks. No ponto B, temos o decaimento do D^0 para o píon e o káon, cujos momentos transversais serão registrados pelo detector CMS.

massa, pseudorapidez, ângulo azimutal). Esse quadrimomento caracteriza o estado físico da partícula no espaço-tempo. Cada componente do quadrimomento está associada a uma propriedade específica; tendo os valores dessas 4 grandezas, podemos construir todas as características cinemáticas destas partículas.

- **Momento transversal (*pT*):** momento linear da partícula na direção perpendicular ao feixe do LHC.
- **Massa (*m*):** massa invariante da partícula; relacionada ao quadrimomento ao quadrado ($P^2 = m^2$, onde usamos velocidade da luz $c = 1$, que é prática comum na física de partículas [5]).
- **Pseudorapidez (η):** escrita em termos do ângulo polar θ em relação à direção do feixe, $\eta = -\ln(\tan(\theta/2))$.
- **Ângulo azimutal (ϕ)**: Define o ângulo no plano transversal ao feixe.

Dessa forma, obtendo essas 4 grandezas para o píon e o káon, teremos o quadrimomento de cada um: P_π (quadrimomento do píon) e P_k (quadrimomento do káon). O quadrimomento do méson D^0 não pode ser obtido diretamente, visto que ele decai muito rapidamente e suas informações cinemáticas não podem ser obtidas diretamente pelo detector CMS. Porém, a partir do quadrimomento de suas filhas, podemos obter o quadrimomento do méson D^0 devido à conservação local de quadrimomento $P_{D^0} = P_k + P_\pi$.

Essa reconstrução é o objetivo deste projeto. Mesmo que o méson D^0 não se desloque suficientemente a fim de ser detectado pelo CMS, podemos construir o quadrimomento das suas filhas e, dessa forma, reconstruir a sua existência. O trabalho aqui expresso busca validar o primeiro passo nessa direção: a obtenção de uma das quatro grandezas do quadrimomento das filhas, o momento transversal, a partir do VAE.

O planejamento para a obtenção dessa grandeza está descrito na seção dedicada à metodologia do trabalho. Conseguindo uma estrutura de VAE que possa obter os momentos transversais, espera-se que o mesmo possa ser feito para η e ϕ (isso não é necessário para a massa, pois os píons e káons possuem massas bem definidas, então basta usar os valores conhecidos).

A importância da reconstrução do méson D^0 reside nos mistérios da pesquisa do QGP. Ele é um estado da matéria extremamente denso e quente (da ordem de trilhões de graus celcius), que se acredita ter existido nos primeiros microssegundos após o Big Bang, quando o universo estava em condições extremas de temperatura e energia. O quark charm que compõe o D^0 carrega algumas características desse plasma, visto que atravessa e “interage” com ele no seu caminho de encontro ao antiquark up. Isso faz com que o próprio D^0, originário desse quark, também carregue essa informação. Em outras palavras, a relevância do D^0 reside na sua capacidade de portar informações sobre as condições do QGP, ou seja, das características iniciais do universo (para entender mais sobre o QGP, a referência [7] pode ser usada). Reconstruir o méson D^0 é crucial para sondar as propriedades excepcionais da matéria quark-glúon, permitindo investigações mais profundas sobre a transição de fase quark-glúon e a dinâmica de partículas em ambientes extremos de temperatura e densidade.

2.2. REDES NEURAIS NO APRENDIZADO DE MÁQUINA

A evolução do aprendizado de máquina trouxe consigo uma importante ferramenta: as redes neurais artificiais. Inspiradas na estrutura do cérebro humano, essas redes são compostas por camadas de informação interconectadas, chamadas de unidades ou nós. O poder das redes neurais reside na capacidade de aprender representações complexas de dados, desde reconhecimento de imagens até processamento de linguagem natural.

A estrutura padrão das redes neurais pode ser representada em 3 divisões: uma

camada de entrada, uma camada “oculta” e uma camada de saída, conforme a Fig. 2.2.

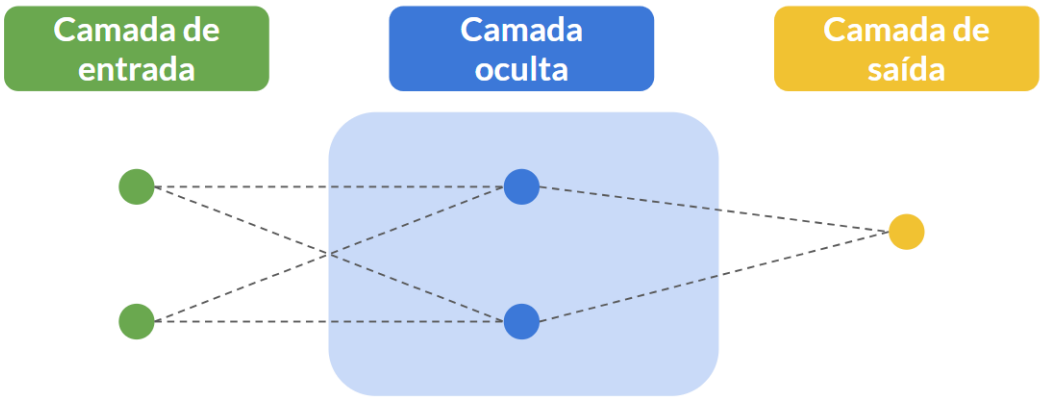

Figura 2.2: Estrutura padrão das redes neurais (adaptado de: [8]).

- **Camada de entrada:** é a primeira camada da rede neural, onde os dados de entrada são inseridos na rede. Cada nó nesta camada representa uma característica ou atributo dos dados de entrada. Esses nós são ativados pelos valores de entrada e transmitem esses sinais para a próxima camada da rede.
- **Camada “oculta”:** estão localizadas entre a camada de entrada e a camada de saída. As camadas ocultas são responsáveis por processar e transformar os sinais de entrada, aprendendo representações mais complexas e abstratas dos dados.
- **Camada de saída:** a camada de saída é a última camada da rede neural, onde os resultados finais ou as previsões são produzidos. Cada neurônio nesta camada representa uma classe, categoria ou valor de saída específico. A ativação dos neurônios nesta camada é usada para gerar a saída final da rede neural.

Além dessa macro divisão em camadas, existem 3 ferramentas essenciais para o entendimento das redes: a função de erro, a retropropagação e os nós.

- **Nós:** representados por círculos na Fig. 2.2, são ferramentas que executam algum tipo de tarefa ou função de processamento em qualquer inserção recebida do nó anterior (ou da camada de entrada). Essencialmente, cada nó contém

uma fórmula matemática que será executada e analisada em cima dos dados recebidos.

- **Retropropagação:** uma ferramenta que garante a eficácia do treinamento da rede. Em termos simples, após cada "passo para frente" da rede neural, a retropropagação realiza um passo para trás, ajustando os parâmetros do modelo de acordo com o erro. Para aprofundamentos no processo de retropropagação, analisar a referência [8].

- **Função de erro:** é uma medida que quantifica o quão distantes as previsões do modelo estão dos valores reais dos dados de treinamento. Normalmente, ela é minimizada para garantir a capacidade do modelo de fazer previsões precisas.

2.2.1. TIPOS DE REDES NEURAIS

Dentro desta arquitetura padrão, existem diversos tipos de redes neurais, que variam em comportamento, tamanho, vantagens e desvantagens. A Fig. 2.3 mostra alguns dos tipos mais comuns de redes neurais.

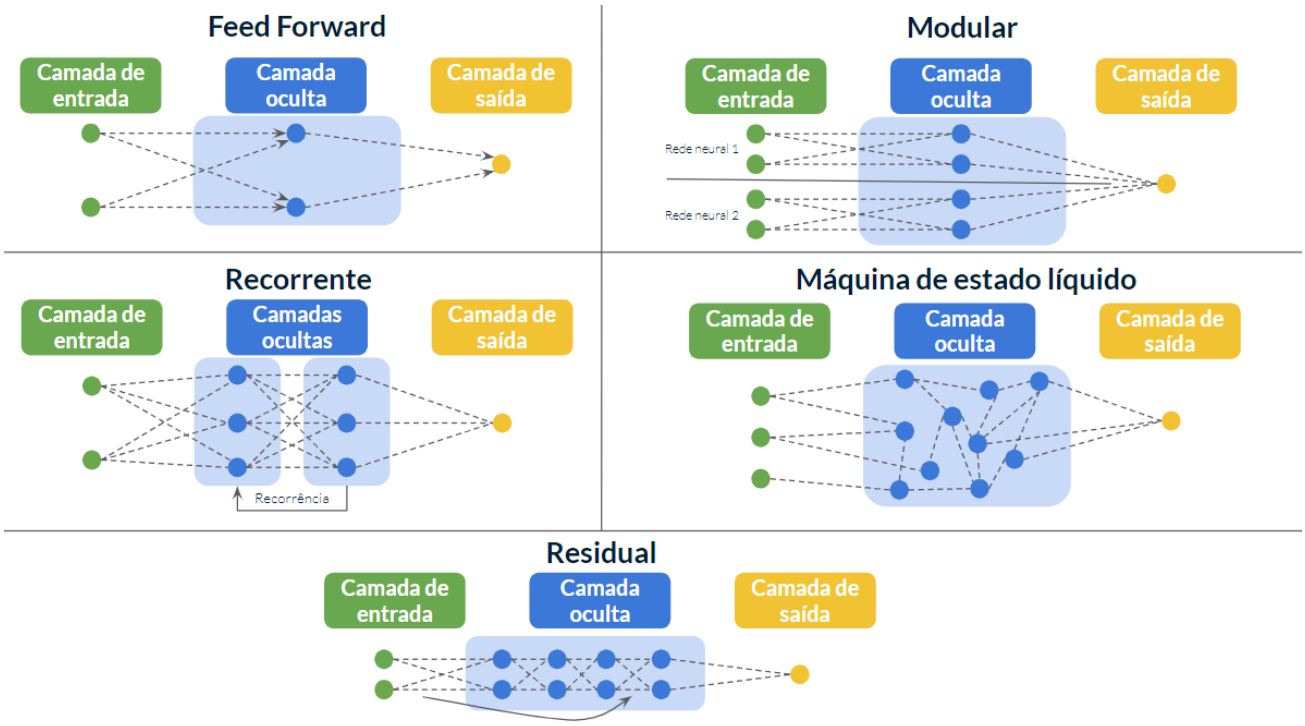

Figura 2.3: Alguns dos tipos mais comuns de redes neurais, seguindo o arquitetura padrão (camada de entrada, camada oculta e camada de saída) (Adaptado de [8]).

- **Rede neural Feed Forward:** essas redes consistem em neurônios organizados em camadas, onde a informação flui em uma direção, da entrada para a saída, sem ciclos ou loops. São amplamente utilizadas em tarefas de classificação, regressão e reconhecimento de padrões em dados estáticos. Seu ponto forte é a capacidade de aprender representações complexas de dados, mesmo sem informações sobre a estrutura temporal dos dados (a ordem e sequência em que os dados são apresentados e processados não é relevante para esse tipo de rede).

- **Rede neural Modular:** essas redes são compostas por várias sub-redes (módulos) interconectadas, cada uma especializada em uma tarefa específica. Essa modularidade facilita a adaptação da rede para diferentes cenários e permite uma interpretação mais intuitiva do seu funcionamento. Sua aplicação indicada é em sistemas complexos que podem ser decompostos em múltiplas tarefas ou subproblemas.

- **Rede neural Recorrente:** essas redes possuem conexões retroativas que permitem que informações sejam transmitidas de volta para camadas anteriores ou para si mesmas ao longo do tempo. Elas são especialmente úteis para modelar sequências temporais, como séries temporais, texto, áudio e vídeo. Sua capacidade de lidar com dados sequenciais as torna ideais para tarefas como tradução automática, reconhecimento de voz e previsão de séries temporais.

- **Rede neural de Máquina de Estado Líquido:** essas redes são inspiradas no funcionamento do cérebro humano e consistem em uma grande quantidade de neurônios altamente interconectados. As informações são processadas dinamicamente em um “estado líquido” de neurônios, onde padrões temporais são formados e reconhecidos. São aplicadas em tarefas de reconhecimento de padrões temporais, como reconhecimento de padrões em sinais temporais complexos e modelagem de sistemas dinâmicos.

- **Rede neural Residual:** essas redes apresentam conexões residuais que saltam uma ou mais camadas, permitindo que informações sejam transmitidas mais facilmente através da rede. Isso ajuda a mitigar problemas de desaparecimento de gradiente e permite treinar redes mais profundas com mais eficiência. São

aplicadas em tarefas onde redes muito profundas são necessárias, como reconhecimento de imagens em alta resolução, segmentação semântica e detecção de objetos.

Mais informações sobre os tipos de redes neurais estão disponíveis na referência [8].

O Autoencoder Variacional, rede neural usada neste projeto, se encaixa na categoria Feed Forward. Isso ocorre porque o VAE possui uma arquitetura composta por camadas de neurônios organizadas em um fluxo direto, onde os dados de entrada são passados através de uma sequência de camadas até a camada de saída sem ciclos ou loops. As principais vantagens desse tipo de rede neural, que justificam a sua escolha para essa aplicação são: simplicidade de arquitetura, eficiência no processo de treinamento e fácil interpretabilidade dos resultados.

2.2.2. FUNÇÕES DE ATIVAÇÃO

Para entender como as redes neurais são capazes de aprender esses conceitos tão complexos, precisa-se evidenciar uma parte muito importante de sua arquitetura: as funções de ativação. Elas são responsáveis por introduzir não-linearidades nas transformações aplicadas às camadas de entrada. Essa capacidade é essencial para lidar com problemas intrinsecamente não-lineares, como reconhecimento de padrões, processamento de linguagem natural e visão computacional.

Ao utilizar funções de ativação, como a ReLU (Rectified Linear Unit), tanh (tangente hiperbólica) ou sigmoid, as redes neurais são capacitadas a melhorar a sua interpretação dos inputs. Elas introduzem não-linearidades nas ativações, permitindo que as redes capturem não apenas correlações simples, mas também padrões intrincados e características hierárquicas nos dados. Em outras palavras, elas operam introduzindo complexidade aos neurônios da rede, aumentando a robustez do aprendizado. A função ReLU, por exemplo, atua ativando o neurônio quando a entrada é positiva e desativando-o para entradas negativas, proporcionando uma resposta não-linear essencial para aprender relações complexas. Um exemplo desse comportamento é exibido na Fig. 2.4.

Dessa forma, mesmo as redes neurais de mais difícil manipulação, com muitas camadas e milhões de parâmetros, têm eficiência em aprender representações

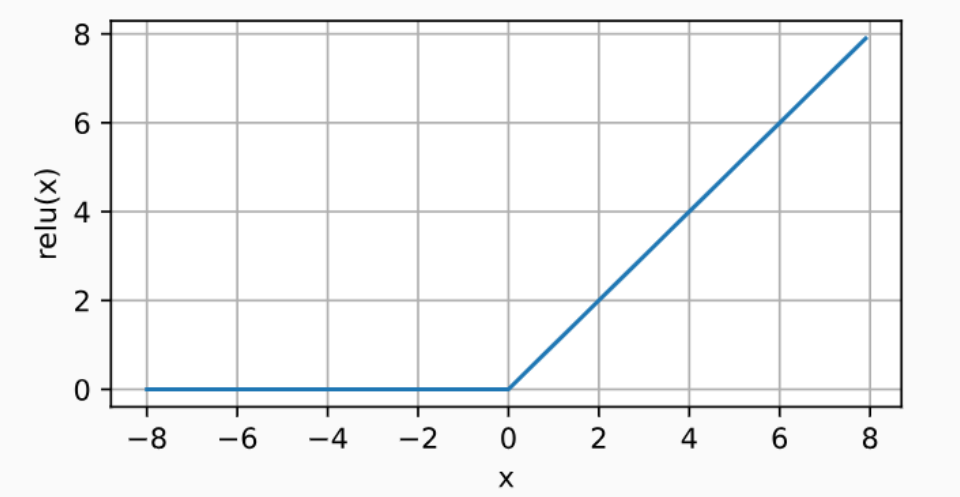

Figura 2.4: Função de ativação ReLU atuando sobre uma função genérica. As entradas negativas são desativadas (fonte: [9]).

hierárquicas de dados brutos, permitindo a extração de características complexas e a resolução de tarefas desafiadoras. Neste trabalho, 3 funções de ativação serão avaliadas: relu, tanh e leaky_relu.

2.2.3. APRENDIZADO SUPERVISIONADO

Dentro do machine learning, duas abordagens amplamente utilizadas são o Aprendizado Supervisionado e o Aprendizado Não Supervisionado (além disso, existem o Aprendizado Semi-supervisionado e o Aprendizado por Reforço, porém esses não são relevantes para o entendimento deste projeto). No aprendizado supervisionado, um modelo é treinado com conjunto de dados rotulados, em que as entradas estão associadas a saídas conhecidas, assim como é representado na Fig. 2.5. O objetivo é que o modelo aprenda a mapear as entradas para as saídas corretas, tornando-se capaz de fazer previsões precisas sobre novos dados de teste. Isso é exemplificado em tarefas como classificação (por exemplo, categorização de e-mails em spam ou não spam) e regressão (por exemplo, previsão de preços de imóveis com base em características).

2.2.4. APRENDIZADO NÃO SUPERVISIONADO

Por outro lado, o aprendizado não supervisionado lida com conjuntos de dados não rotulados, em que o modelo deve encontrar estrutura nos dados sem orientação

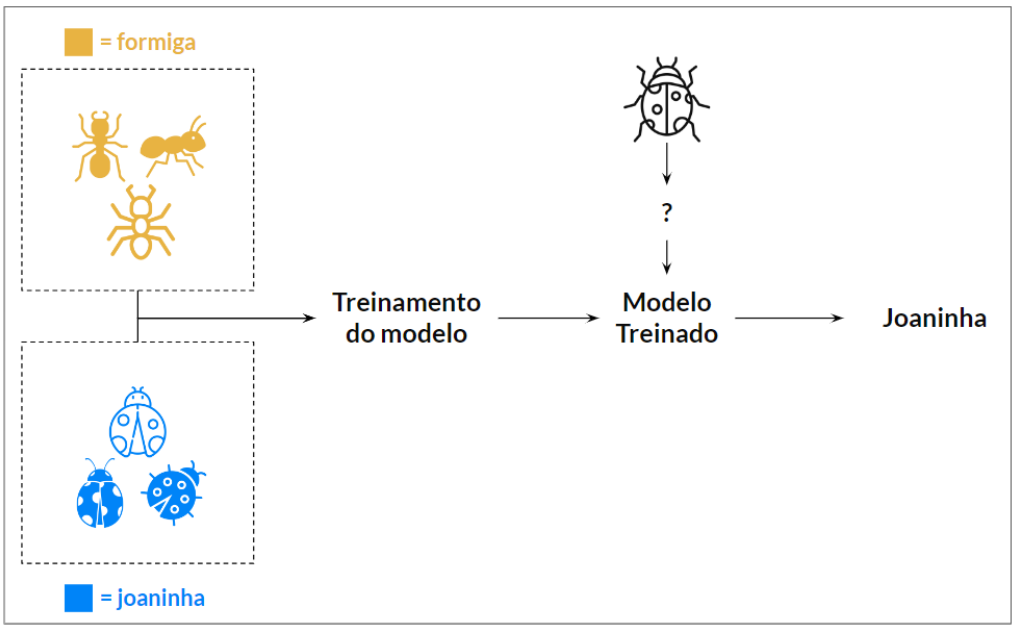

Figura 2.5: Modelo supervisionado, onde as entradas possuem legendas para suportar a aprendizagem do modelo: treinado com dados legendados (adaptado de: [10]).

externa, como exposto na Fig. 2.6. Isso pode envolver a identificação de agrupamentos naturais em dados (clustering) ou a redução de dimensionalidade para simplificar a representação dos dados, entre outras tarefas. O aprendizado não supervisionado é fundamental para explorar padrões intrínsecos e descobrir informações ocultas em grandes volumes de dados.

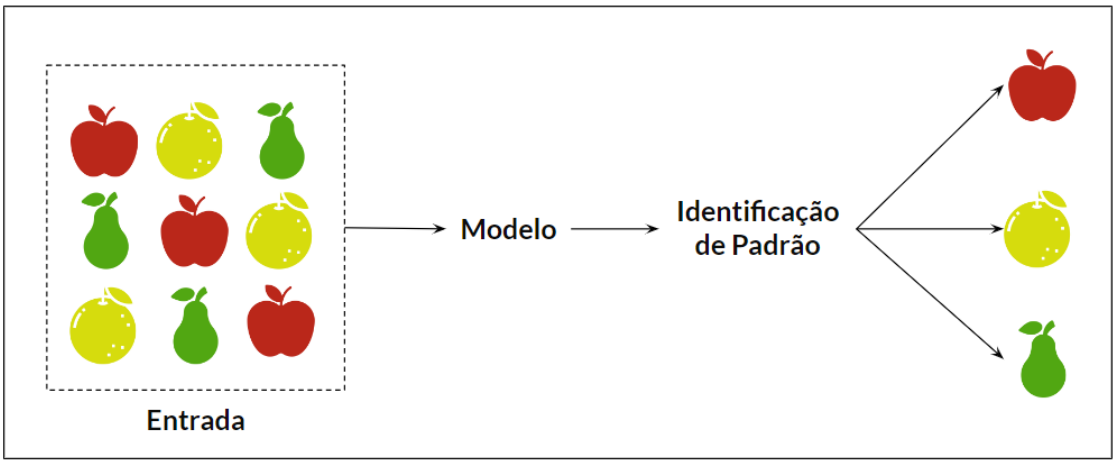

Figura 2.6: Modelo não supervisionado, em que os dados de entrada não têm legenda, portando a aprendizagem ocorre apenas pela identificação de padrões nos dados (adaptado de: [11]).

2.2.5. MODELOS GENERATIVOS NO APRENDIZADO DE MÁQUINA

Além do aprendizado supervisionado e não supervisionado, um desenvolvimento notável no campo do machine learning é a ascensão dos Modelos Generativos. Estes modelos são projetados para criar novos dados que são semanticamente semelhantes aos dados de treinamento, permitindo a geração de exemplos que parecem ter sido amostrados do conjunto de dados original. Dentre as vantagens deste output, destaca-se a possibilidade de geração de dados para simular situações reais em contextos onde a obtenção de novos dados é difícil ou custosa, aprimorando assim a eficiência e a eficácia de várias práticas de machine learning e análise de dados (na Fig. 2.7 ilustramos uma comparação com modelos de machine learning tradicionais). Modelos generativos, como as redes Generativas Adversariais (GANs) e as redes Autoregressivas, são exemplos de modelos generativos que têm essa habilidade. Os modelos generativos se tornam uma ferramenta poderosa e que podem ser testadas para satisfazer a proposta deste projeto.

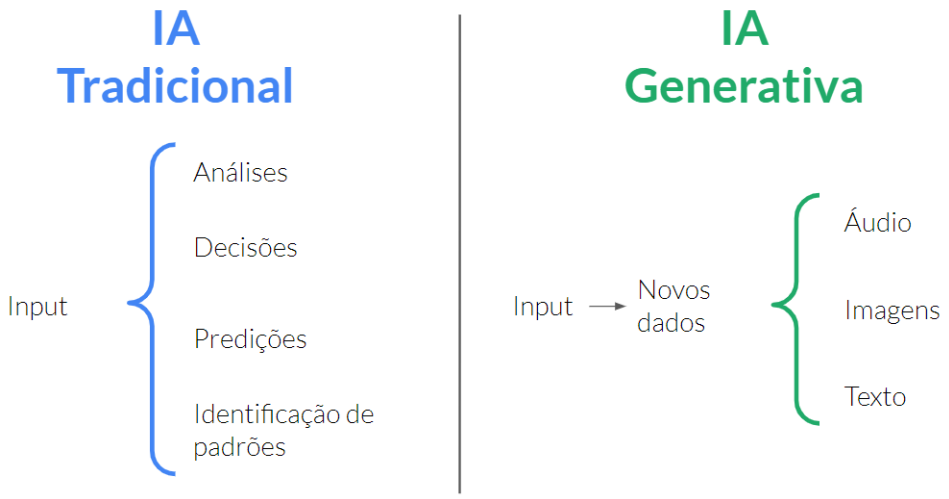

Figura 2.7: Comparação das características da IA tradicional e da IA generativa.

2.3. AUTOENCODER

Os autoencoders são uma classe de rede neural artificial não supervisionada que desempenham um papel crucial no campo do aprendizado de máquina e são amplamente utilizados em tarefas de redução de dimensionalidade, redução de ruído e

detecção de anomalias. Elas são do tipo feed forward, e têm, portanto, uma estrutura intrincada e que pode ser aplicada de várias maneiras para aprender representações eficazes dos dados, permitindo a extração de informações valiosas e ocultas. Um autoencoder é composto por três partes principais: o encoder (codificador), o vetor latente (ou espaço latente) e o decoder (decodificador).

- **Encoder**: primeira parte do Autoencoder é responsável por mapear os dados de entrada para um espaço de menor dimensão, conforme a Fig. 2.8. Nesta etapa, as informações essenciais são extraídas do input e comprimidas em um vetor de características, também conhecido como vetor latente.

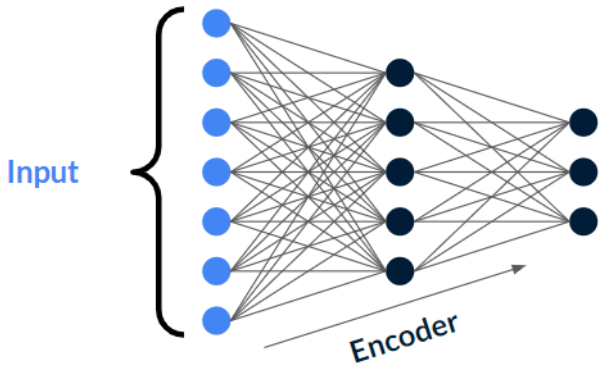

Figura 2.8: Encoder mapeando os dados de entrada para uma menor dimensão.

A tarefa do encoder pode ser visualizada facilmente no seguinte exemplo: um conjunto de dados de entrada composto por pontos no plano cartesiano descreve cada ponto em 3 dimensões: localização do ponto de acordo com os eixos X, Y e Z. Porém, ao plotar um gráfico desses dados, percebe-se que os pontos descrevem uma espiral perfeitamente, ver Fig. 2.9. Nesse caso, pode-se diminuir a dimensão da representação de cada ponto, pois precisa-se apenas de uma grandeza para definir todos os diferentes dados desse conjunto: o ângulo de desvio do ponto com relação à origem do sistema.

Em outras palavras, é uma situação em que se pode tomar vantagem da estrutura dos dados no sistema para mapeá-los para um sistema de coordenadas de

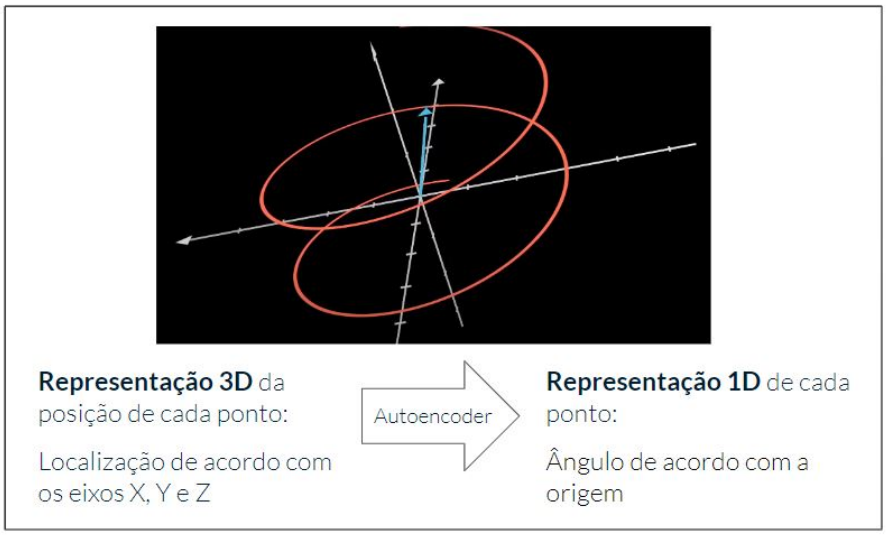

Figura 2.9: Gráfico representando um conjunto de pontos. Pode-se perceber que esses dados têm uma estrutura (espiral), habilitando a diminuição de dimensionalidade.

dimensão inferior. Não são mais necessárias 3 coordenadas para descrever cada ponto, mas apenas uma. Essa é uma ilustração da função do encoder.

- **Vetor Latente**: é uma representação de dimensionalidade reduzida dos dados de entrada. Representa a informação extraída do encoder e contém características que resumem as informações mais importantes contidas nos dados originais, como apresentado na Fig. 2.10. Este vetor é a representação compacta dos dados. No exemplo da espiral acima, o vetor latente seria a representação 1D de cada ponto.

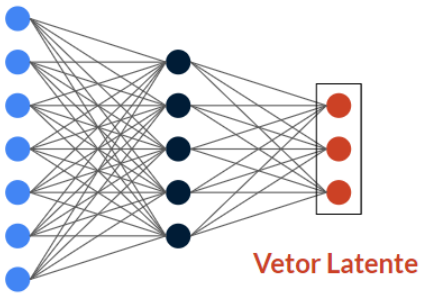

Figura 2.10: Representação reduzida dos dados de entrada.

- **Decoder**: realiza a tarefa inversa do encoder. Ele mapeia o vetor latente de volta para o espaço de tamanho original, tentando reconstruir os dados de entrada da

maneira mais fiel possível (Fig. 2.11). Ele é, portanto, a parte final do autoencoder, e tem como objetivo recompor uma aproximação precisa dos dados originais.

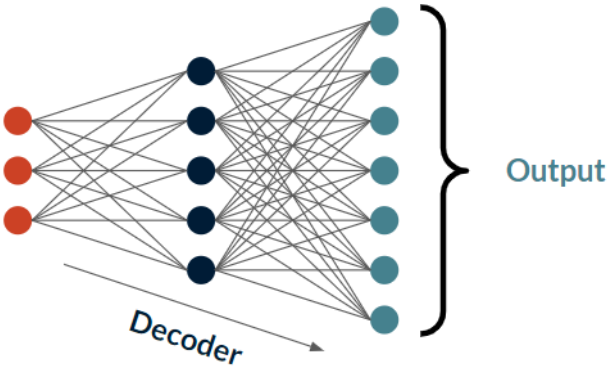

Figura 2.11: Decoder construindo a saída de volta à dimensão original.

Portanto, o autoencoder pode ser compreendido como um sistema que recebe um input, comprime-o em uma representação de menor escala e depois o reconstrói da maneira mais fiel possível, conforme a Fig. 2.12. Dado esse cenário, pode-se levantar a questão de qual é a verdadeira utilidade desse processo, já que se procura um output tão próximo do input quanto possível. É diante desse questionamento que surgem as principais aplicações dos autoencoders, conhecidas como denoising e anomaly detection.

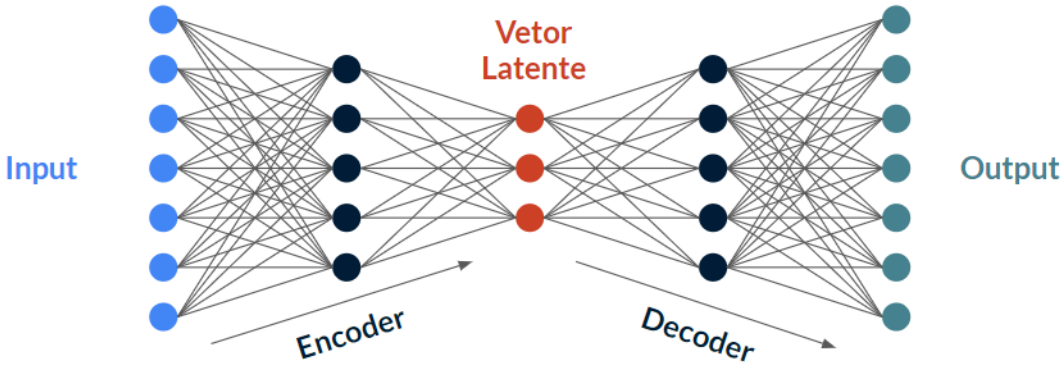

Figura 2.12: Autoencoder completo.

2.3.1. DENOISING

Um autoencoder treinado perante a metodologia de denoising é projetado para lidar com dados ruidosos ou incompletos. Esse processo baseia-se em alimentar o modelo com dados “manipulados”, com ruído propositalmente inserido, de maneira que ele entenda que a distorção não faz parte da estrutura padrão do input. Esse processo está sendo realizado na Fig. 2.13. A vantagem, nesse caso, é que agora temos um sistema que sabe separar sinal de ruído, limpando os dados de entrada “reais” que normalmente possuem imperfeições.

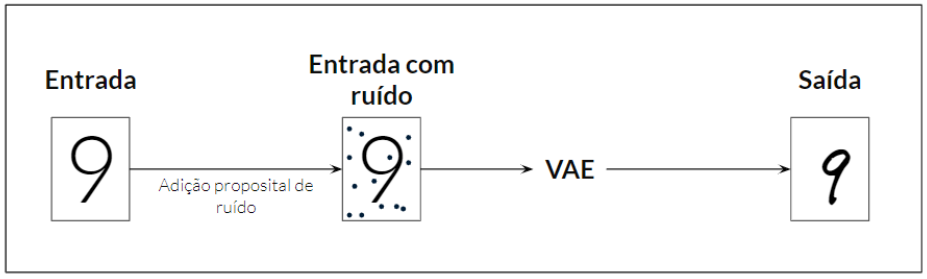

Figura 2.13: Treinamento do autoencoder com ruído propositalmente adicionado à entrada (adaptado de: [12]).

2.3.2. ANOMALY DETECTION

Como os autoencoders são treinados para reconstruir dados com alta precisão, eles podem ser usados para identificar anomalias que não podem ser reconstruídas com eficácia. Quando uma entrada é significativamente diferente da sua reconstrução, isso pode ser um indicativo de uma anomalia. Portanto, autoencoders são eficazes na detecção de padrões incomuns ou comportamento anômalo em dados, o que é crucial em muitos domínios, incluindo segurança cibernética, manutenção preditiva e controle de qualidade.

2.3.3. FUNÇÃO DE PERDA

A função de perda, também conhecida como erro de reconstrução, é um componente crítico no treinamento de autoencoders. Essa função avalia o quão bem o modelo é capaz de reconstruir a entrada a partir do vetor latente, sendo definida como a

diferença entre entrada (input) e reconstrução (output). Minimizando a função de perda, estamos incentivando o autoencoder a encontrar a melhor representação latente possível, tornando a saída extremamente assertiva. Uma função de perda comum é o mean squared error (MSE), que calcula a média dos quadrados das diferenças entre os valores de entrada e suas reconstruções. No caso abaixo, N é o número total de elementos na entrada, enquanto x e $\hat{x}$ são a entrada e a sua reconstrução, respectivamente:

$$L(x, \hat{x}) = \frac{1}{N} \sum_{i=1}^{N} (x_i - \hat{x}_i)^2 \tag{2.1}$$

2.4. AUTOENCODER VARIACIONAL (VAE)

Através das funcionalidades do autoencoder, podemos perceber que ele realmente é um método promissor e repleto de aplicações. No entando, existem algumas limitações na parte generativa desse sistema que demandam uma evolução na sua arquitetura. Para visualizar isso, basta isolar o decoder, a parte generativa do processo. Ele seguirá utilizando o vetor latente para gerar o output, porém, isso só é possível quando se sabe exatamente quais valores devem compor esse vetor para termos uma saída válida. Por exemplo, se temos um autoencoder treinado para gerar diversas fotos de Albert Einstein, Marie Curie e Grace Hopper, a parte generativa dele conseguirá criar uma foto representando essas pessoas apenas se lhe for informado exatamente onde e quais são os valores que geram esses outputs através do vetor latente. Porém, não há como confirmar essas informações. Como apresentado na Fig. 2.14, os valores que irão compor essas imagens funcionam como “piscinas”, isto é, regiões do espaço multidimensional em que se define o vetor latente.

Cada piscina é uma distribuição de vetores latentes que nos darão aquela imagem ao alimentarmos o decoder. Porém, se selecionarmos um valor qualquer nesse espaço, temos uma probabilidade muito grande de “errar” todas as piscinas e gerar um output sem sentido algum. Nesse sentido, é necessário que se tenha uma maneira conveniente de escolher esses valores corretamente, ou seja, de “acertar” a piscina desejada. Uma maneira muito efetiva de garantir isso é restringindo os valores de vetores latentes para uma região contínua específica (conhecida), conforme a Fig. 2.15.

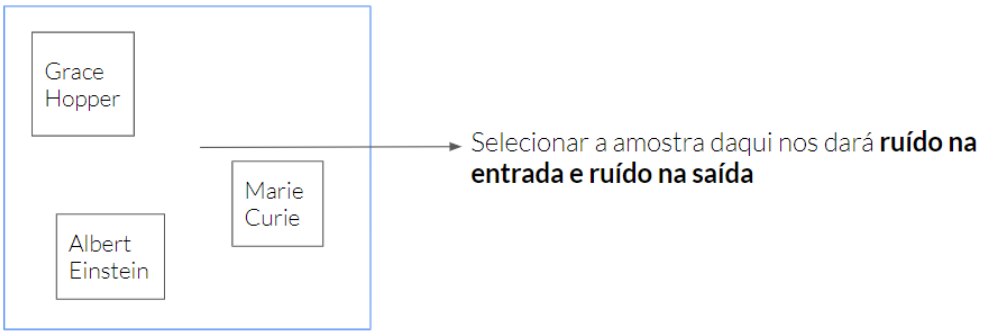

Figura 2.14: Piscinas dos vetores que dão as respectivas imagens como saída. As interseções entre as piscinas resultam em um resultado aproximado entre os dois resultados.

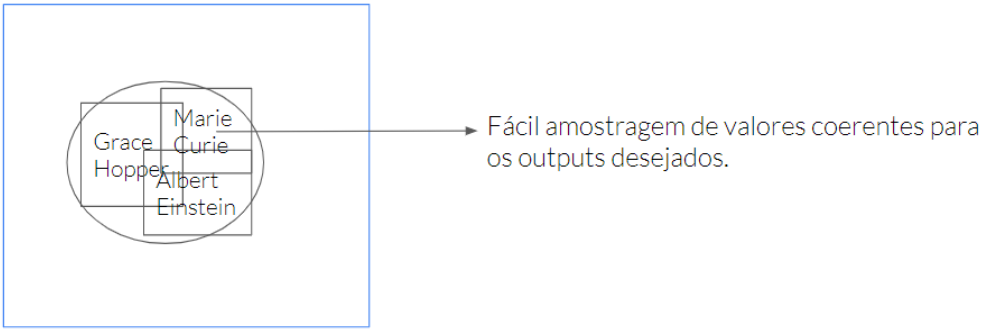

Figura 2.15: Junção dos vetores em uma distribuição conhecida, facilitando a amostragem.

Dessa forma, poderemos amostrar vetores desse espaço à vontade, obtendo os outputs desejados na medida em que variamos a entrada. É aqui que o VAE é introduzido. Sua arquitetura foi pensada para resolver esse complexo impasse na parte generativa do autoencoder e possibilitar a restrição dos vetores numa região conhecida.

2.4.1. O DIFERENCIAL DO VAE

Ao invés de mapear o input para um vetor latente fixo, como faz o autoencoder, o VAE mapeia os dados de entrada para uma distribuição! Isso ocorre através da função de perda, como será apresentado a seguir. A partir disso, podemos amostrar facilmente desse espaço contínuo e conhecido. Tipicamente, a distribuição utilizada é a normal, visto que ela pode ser facilmente definida através da média e do desvio padrão. Ter essa distribuição criada permite que o VAE use mais um “truque”: mapear o output do encoder para a média e o desvio padrão dela, ao invés de passar toda a informação diretamente para o vetor latente, como faz o autoencoder. Dessa forma, a arquitetura do VAE mantém a tríade de componentes dos autoencoders normais (encoder, vetor latente e decoder), porém, adicionando dois vetores previamente à representação latente, conforme a Fig. 2.16. A partir disso, o vetor latente (z) é construído com uma amostragem da média e uma amostragem do desvio padrão, $\vec{z} = \mu + \sigma$, fazendo com que se tenha apenas duas variáveis para serem treinadas, e não uma distribuição inteira, tornando esse processo finalmente possível.

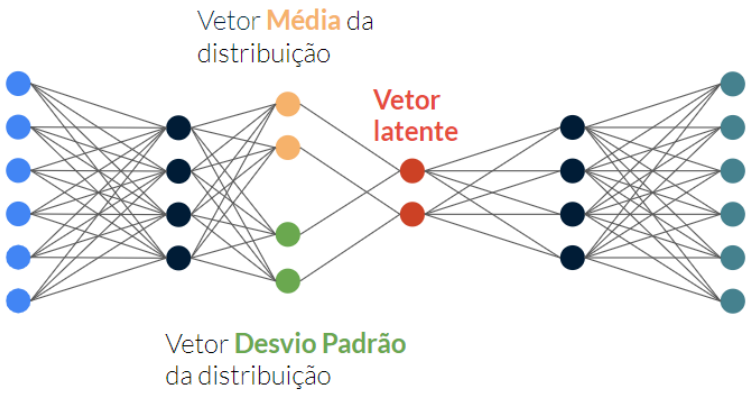

Figura 2.16: VAE completo, com a saída do encoder mapeada para vetores de média e desvio padrão anteriormente à concepção do vetor latente.

2.4.2. FUNÇÃO DE PERDA DO VAE

A função de perda em VAEs desempenha um papel crucial na estruturação do treinamento e na geração de representações latentes significativas. Para compreender a formulação desta função, é essencial abordar os objetivos do VAE: reconstrução precisa dos dados de entrada, da mesma forma que o autoencoder comum, e a alocação dos vetores latentes em uma distribuição de referência, para que possamos amostrar de um local conhecido, conforme apresentado anteriormente. Para o entendimento deste projeto, pode-se descrever essa função de maneira reduzida, representando-a apenas como uma composição de dois termos principais: o erro de reconstrução e a divergência Kullback–Leibler (KLD):

$$\mathcal{L} = L_{\text{rec}} + D_{\text{KL}}(E \| Z),$$

Em que E representa a distribuição aproximada do espaço latente e D_{KL} denota a Divergência Kullback–Leibler entre E e a distribuição padrão $Z \approx \mathcal{N}(0, 1)$ (onde $\mathcal{N}(0, 1)$ é a distribuição normal de média 0 e desvio padrão 1). Conforme será apresentado na Seção 2.5.3, essa função permitirá a realização de ajustes nos parâmetros do VAE, permitindo adequar o output para o formato desejado. Para entender mais sobre a função de erro do VAE, ver Ref. [4].

- **Erro de Reconstrução**: O primeiro termo da função de perda quantifica a precisão com que o VAE reconstrói os dados de entrada a partir da representação latente. Minimizar este termo é crucial para assegurar que a geração do VAE produza reconstruções precisas e que o modelo aprenda representações latentes que sejam informativas e úteis para a reconstrução.

- **Divergência KL**: Durante o treinamento, o VAE irá “aprender” como os dados estão distribuídos. A partir disso, o segundo termo da função de perda, a divergência Kullback–Leibler (KLD), é introduzida. A divergência KL quantifica a discrepância entre duas distribuições de dados. Ela é muito empregada em problemas de aprendizado não supervisionado, pois pode ser utilizada para medir a dissimilaridade entre distribuições de dados observados e gerados. Nesse caso, iremos usar a KLD para calcular a diferença entre essa distribuição

"aprendida" pelo VAE e a distribuição normal padronizada (gaussiana com média 0 e desvio padrão 1). Dessa forma, minimizar esse termo regulariza a distribuição latente, visto que menores valores de KLD significam representações cada vez mais próximas da gaussiana, um espaço conhecido e facilmente amostrável. Um exemplo do funcionamento da KLD está na Fig. 2.17

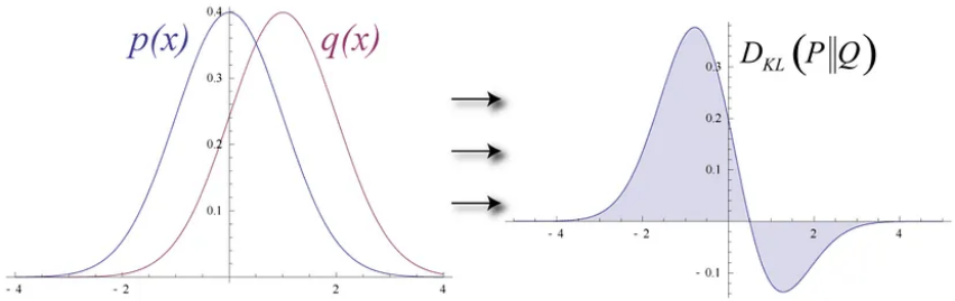

Figura 2.17: Divergência KL de duas gaussianas $p(x)$ e $q(x)$ (fonte: [13]).

Minimizando a função de erro durante o treinamento aumentamos as chances de uma reconstrução assertiva dos dados de entrada e possibilitando a amostragem para a parte generativa do VAE a partir da distribuição desejada.

2.4.3. REPARAMETRIZAÇÃO

A próxima ferramenta que explica o funcionamento dos autoencoders variacionais surge a partir de uma problemática que se deve resolver antes de iniciar o treinamento do VAE: como poderemos realizar a retropropagação eficientemente em presença de variáveis latentes estocásticas?

No treinamento de redes neurais convencionais, a retropropagação é direta e eficaz. Ela segue o princípio da descida do gradiente, em que os gradientes da função de perda são calculados e utilizados para minimizar a função de perda, como explicado na Seção 2.2. Em redes neurais convencionais, isso ocorre sem impeditivos, visto que todas as operações são diferenciáveis.

No entanto, nos VAEs, a amostragem aleatória da média e do desvio padrão, como explicado na Seção 2.4.1, introduz a presença de variáveis latentes estocásticas. A amostragem direta dessas variáveis durante a retropropagação não é diferenciável, o

que impede o cálculo dos gradientes necessários para ajustar os parâmetros do modelo. Ou seja, a amostragem aleatória de valores de média e desvio padrão para o vetor latente impede que a retropropagação minimize a função de perda.

A reparametrização é uma técnica que contorna esse problema ao expressar a amostragem estocástica como uma operação determinística e diferenciável. Ela funciona introduzindo uma variável aleatória (ϵ) que segue uma distribuição conhecida (geralmente uma gaussiana padrão) e utilizando-a para realizar a amostragem estocástica. Essa amostragem é então combinada com os parâmetros do encoder (média e desvio padrão) para gerar o vetor latente de maneira diferenciável: $z = \mu + \sigma \odot \epsilon$, onde $\epsilon \approx N(0,1)$, μ e σ são a média e o desvio padrão, respectivamente, ϵ é a variável aleatória amostrada, e $\odot$ denota a multiplicação elemento a elemento.

Como pode ser observado, ϵ é multiplicada apenas por σ e não por μ. Isso ocorre porque a média já é utilizada para representar a posição central da distribuição latente, e adicionar um componente estocástico à média não seria apropriado para a modelagem dos dados. Isso permite que a amostragem do vetor latente seja condicionada às estatísticas conhecidas. Ao multiplicar ϵ por σ, essencialmente, ajusta-se a dispersão da distribuição latente em torno da média, controlando a amplitude das amostras geradas.

Essa formulação permite que a retropropagação ocorra sem problemas através do vetor latente, pois a amostragem é agora uma operação determinística com relação aos parâmetros do modelo, tornando possível o treinamento desses modelos generativos probabilísticos. Um resumo da transformação do vetor latente Z através da reparametrização está na Fig. 2.18.

2.4.4. HIPERPARÂMETROS

Os hiperparâmetros do VAE são parâmetros que não são aprendidos diretamente pelo modelo durante o treinamento, mas que precisam ser definidos antes do processo de treinamento. Ajustar esses hiperparâmetros é crucial para o projeto porque eles influenciam diretamente a capacidade do modelo de capturar as características dos dados de entrada e gerar saídas de alta qualidade. Eles funcionam como "engrenagens" que devem ser ajustadas para o VAE gerar o output desejado. Um bom

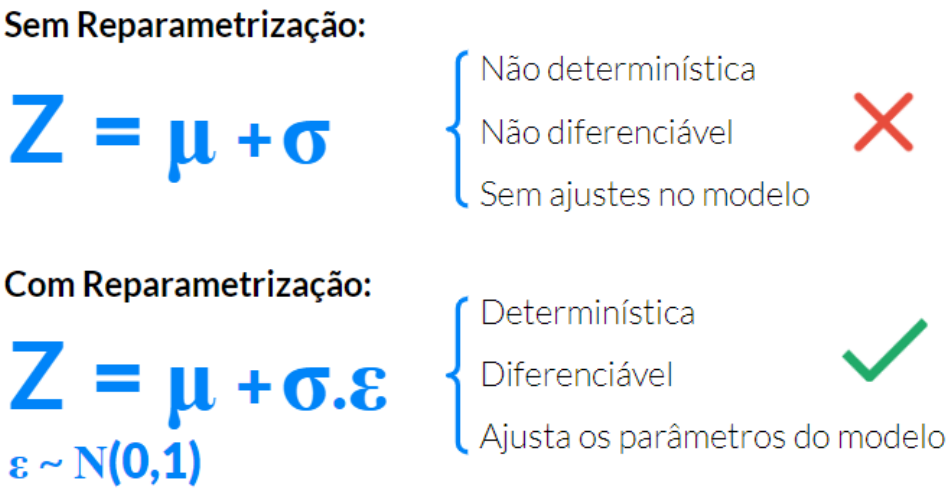

Figura 2.18: Resumo das descrições do vetor latente Z sem e com a técnica de reparametrização.

ajuste pode melhorar significativamente o desempenho do modelo, resultando em melhores reconstruções e gerações de dados que são mais semelhantes aos dados reais. O VAE deste projeto possui 7 hiperparâmetros:

- **Número de camadas ocultas no VAE**: Define a profundidade da rede neural, influenciando a capacidade do modelo de aprender representações complexas dos dados;
- **Dimensão das camadas ocultas**: Controla a largura de cada camada, afetando a quantidade de informação que pode ser processada e armazenada em cada nível da rede;
- **Dimensão do vetor latente**: Determina o tamanho do espaço onde os dados são comprimidos, afetando a capacidade do VAE de capturar variações nos dados de entrada;
- **Tamanho do batch**: Define o número de amostras que o modelo processa antes de atualizar seus pesos, afetando a estabilidade da estimativa do gradiente;
- **Taxa de aprendizado**: Controla a velocidade com que o modelo ajusta seus pesos durante o treinamento, influenciando a estabilidade e a eficiência da convergência;
- **Função de ativação utilizada**: Determina qual será a função de ativação utilizada no VAE;

- **Otimizador utilizado**: Seleciona o algoritmo que ajusta os pesos do modelo, impactando a eficiência e a velocidade do processo de treinamento.

Como será demonstrado na Seção 4.1, encontrar o melhor valor para um hiperparâmetro é uma tarefa difícil e demorada, principalmente se for feita manualmente. Cada hiperparâmetro pode ter um impacto significativo no desempenho do modelo, e encontrar a combinação ideal entre eles geralmente requer testar muitas configurações diferentes.

2.5. FERRAMENTAS ADICIONAIS DO VAE

Além das ferramentas-padrão que compõem a arquitetura do VAE, existem funcionalidades adicionais que podem ser inseridas na estrutura para melhorar a sua performance. Essas ferramentas não são partes obrigatórias do VAE, mas podem ser testadas em conjunto com a estrutura padrão para aumentar a eficácia e a aplicabilidade do VAE para a problemática deste projeto. Nesta seção, será apresentada a biblioteca Optuna, uma poderosa ferramenta para a otimização de hiperparâmetros; a desnormalização dos dados, uma técnica essencial para aumentar a estabilidade do treino; e o Beta-VAE, uma extensão que permite um controle mais refinado entre a regularização do espaço latente e a precisão da reconstrução.

2.5.1. BIBLIOTECA OPTUNA

A partir do desafio de ajustar os hiperparâmetros do VAE de maneira manual perante milhões de combinações possíveis, torna-se clara a necessidade de automatizar esse processo. Para essa finalidade, uma ferramenta comum adicionada à arquitetura dos autoencoders variacionais é a biblioteca Optuna.

O Optuna é uma biblioteca de otimização de hiperparâmetros. Ela é utilizada principalmente para realizar essa tarefa em redes neurais. Esse processo se torna valioso e necessário em redes neurais de alta complexidade, ou seja, que precisam de ajustes em diversos parâmetros para otimizar o output. Como o VAE deste projeto usa 7 hiperparâmetros (listados na subseção anterior), o Optuna é um excelente candidato para a sua arquitetura.

O Optuna opera através da definição de uma função objetivo que inclui o processo de treinamento e validação do VAE. Ele tentará atingir esse objetivo através de múltiplos

ensaios, chamados de trials. Durante cada trial, o Optuna sugere valores para os hiperparâmetros, treina o VAE com esses valores, e avalia o desempenho do modelo através da perda de validação. A biblioteca então ajusta os hiperparâmetros em trials subsequentes com base nos resultados anteriores, buscando minimizar a perda de validação e encontrar a combinação de hiperparâmetros que proporciona o melhor desempenho.

Um exemplo básico de funcionamento do Optuna está na Fig. 2.19. Os trials estão representados nas colunas, e os hiperparâmetros nas linhas. O Trial 1 seleciona valores aleatórios para iniciar a otimização, e um erro de validação é gerado a partir deste primeiro passo. Nos trials seguintes, o Optuna testa novos valores com base nos erros de validação anteriores. Tomando o primeiro hiperparâmetro como exemplo, pode-se observar que o Optuna começou com o valor 2, e obteve erro de validação 19. Ele testou subir o valor para 5, e obteve erro de validação 21. Esse aumento sinaliza que o ajuste foi “incorreto” ou “para o lado errado”. Dessa forma, nos trials subsequentes, o Optuna continua o ajuste e entende que o melhor valor para o hiperparâmetro 1 é 3. Neste exemplo, ao final dos seis trials, os valores selecionados para os hiperparâmetros serão os presentes no trial 5, visto que ele teve o menor erro de validação.

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Trial 6
Hiperparâmetro 1	2	5	4	3	3	3
Hiperparâmetro 2	330	360	280	270	273	275
Hiperparâmetro 3	0,08	0,09	0,08	0,07	0,08	0,07
Hiperparâmetro 4	A	D	F	B	B	C
Erro de Validaçao	19	21	16	12	7	9

Figura 2.19: Exemplo de funcionamento dos trials da biblioteca Optuna.

Para executar a otimização dos hiperparâmetros, o Optuna requer 2 inputs importantes do usuário:

- **Número de trials**: quantos trials serão executados para otimizar os

hiperparâmetros. Essa escolha deve ser cuidadosa, já que um número muito baixo de trials não será o suficiente para que o Optuna encontre bons resultados, e um número muito alto torna o processo demorado e ineficiente;

- **Banda de valores para cada hiperparâmetro**: intervalo de valores coerentes para cada hiperparâmetro, que serão utilizados como balizadores mínimo e máximo nos trials. Por exemplo, para o número de camadas, o mínimo valor coerente para uma arquitetura padrão de VAE é 2, e o máximo é 5. A chance do melhor valor para esse hiperparâmetro estar fora deste range é quase nula, portanto limita-se o espaço de procura do Optuna neste intervalo. As bandas utilizadas para o VAE deste projeto serão apresentadas na Seção 4.2.

A Fig. 2.20 demonstra uma execução real dos trials do Optuna. O erro de validação está diminuindo com o decorrer dos trials. O trial com o menor erro de validação foi o 10. Pode-se perceber que a diminuição do erro de validação é mais expressiva nos primeiros trials, e que atinge um “platô” nos trials finais, onde o erro de validação quase não muda mais. Por esse motivo, aumentar drasticamente o número de trials não irá melhorar o output do VAE na mesma proporção.

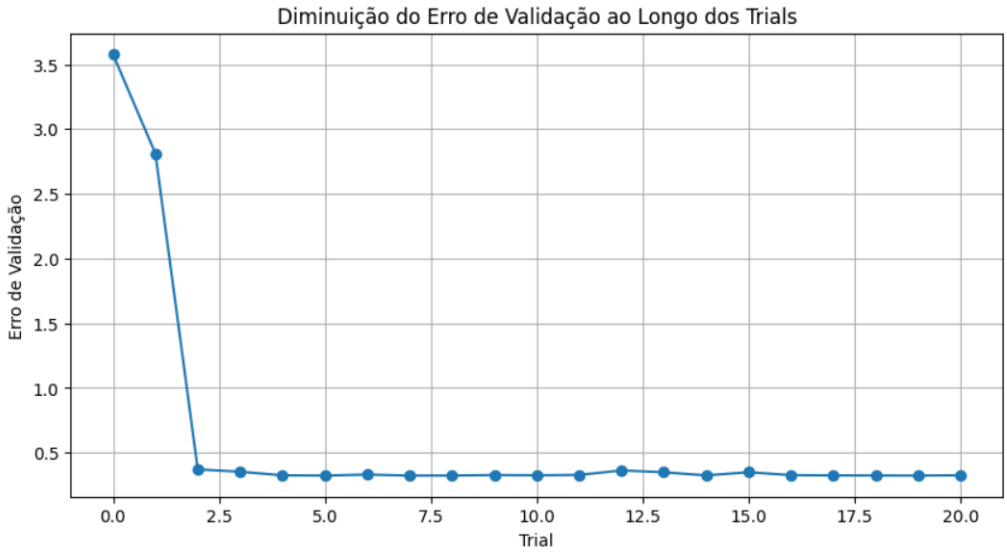

Figura 2.20: Exemplo de execução dos trials da biblioteca Optuna.

É importante lembrar que o Optuna não é uma ferramenta que testa todas as combinações possíveis de valores para os hiperparâmetros. Isso seria algo ineficiente e com tempo altíssimo de processamento.

Utilizando o Optuna, o processo de ajuste de hiperparâmetros torna-se automatizado e mais eficiente, permitindo que o VAE aumente seu potencial de desempenho com menos esforço manual e em menos tempo. Os resultados da aplicação do Optuna serão abordados na Seção 4.2.

2.5.2. NORMALIZAÇÃO DOS DADOS DE ENTRADA

A normalização dos dados de entrada é uma prática altamente recomendada e amplamente adotada no quesito de otimização das redes neurais. Ao normalizar os dados, eles estão sendo essencialmente ajustados para que possuam uma escala comum, geralmente com média zero e desvio padrão unitário. Esse processo pode ajudar a performance do VAE, por 2 razões principais:

- **Estabilização do Treinamento**: A normalização pode reduzir a variabilidade nos dados de entrada, ajudando a tornar o processo de treinamento mais estável. Dados com diferentes escalas podem causar oscilações nos gradientes durante o treinamento, dificultando a convergência do modelo. Ao normalizar os dados, garantimos que todas as características contribuem de maneira equilibrada para o treinamento, evitando que algumas características dominem o aprendizado.
- **Uniformidade nas Atualizações dos Pesos**: Quando os dados são normalizados, os gradientes calculados durante a retropropagação (introduzida na Seção 2.2) têm uma magnitude mais uniforme, resultando em atualizações de pesos mais consistentes e previsíveis. Isso é particularmente importante para redes neurais profundas como o VAE, onde variações grandes nos gradientes podem causar problemas de explosão ou desaparecimento de gradientes, prejudicando o treinamento. Com dados normalizados, cada peso na rede recebe atualizações proporcionais e equilibradas, melhorando a performance geral do VAE.

Existem outros benefícios em normalizar os dados de entrada em redes neurais, os quais estão descritos na Ref. [14], porém esses dois são os principais para este projeto.

A normalização dos dados envolve ajustar os valores dos dados de entrada para que eles tenham uma média específica e uma variância constante. Isso é feito através da seguinte fórmula:

$$X_{\text{normalizado}} = \frac{X - \text{média(X)}}{\text{desvpad}(X)}$$

Onde:

- X é o valor original dos dados;
- média(X) é a média dos dados;
- desvpad(X) é o desvio padrão dos dados.

A Fig. 2.21 demonstra uma distribuição semelhante aos dados de entrada do projeto, os quais serão detalhados na Seção 3.1, em seu formato original e no formato normalizado. Após a normalização, os dados estão centrados em zero, e o desvio padrão se aproxima de 1. A distribuição de baixo é mais homogênea e contém menos variabilidade, oferecendo um treinamento mais estável para o VAE quando comparada com a distribuição de cima.

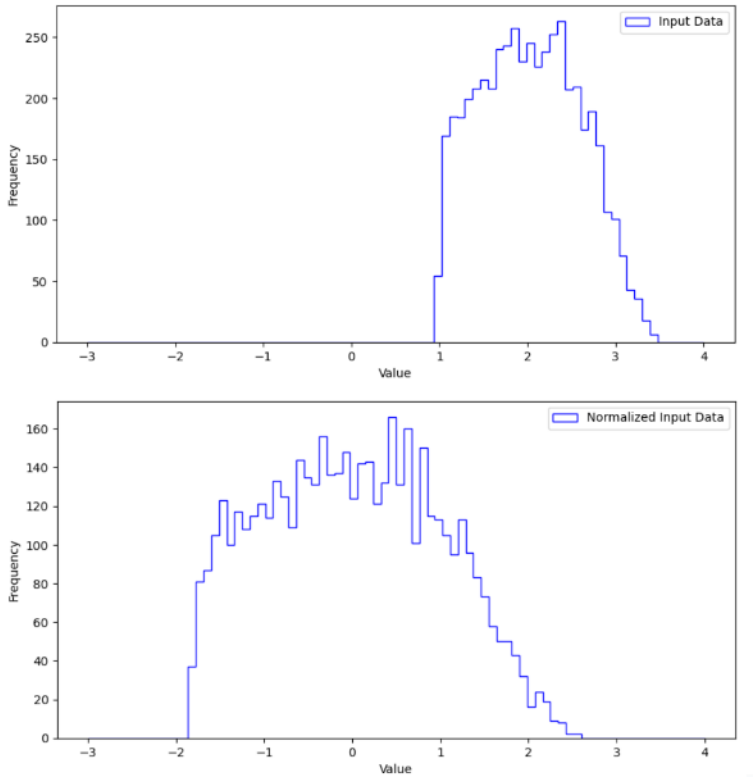

Figura 2.21: Dados de entrada originais (acima) e normalizados (abaixo), usando a mesma escala no eixo horizontal.

Após a adição da normalização dos dados, o output do VAE também estará normalizado, e pode ser desnormalizado através do processo inverso:

$$X_{\text{original}} = X_{\text{normalizado}} \times \text{desvpad}(X) + \text{média}(X)$$

A normalização ajuda a garantir que o VAE funcione de maneira mais estável e eficiente, melhorando a qualidade das amostras geradas e tornando o processo de treinamento mais robusto. Os resultados da aplicação da normalização dos dados de entrada serão abordados na Seção 4.3.

2.5.3. BETA-VAE

Outra ferramenta que pode ser adicionada à arquitetura do VAE para melhorar a sua performance é o fator beta ($0 \leq \beta \leq 1$) treinamento do VAE para ajustar a influência relativa entre os termos. Conforme apresentado na Seção 2.4.2, a função de perda $\mathcal{L}$ é descrita por:

$$\mathcal{L} = L_{\text{rec}} + D_{\text{KL}}(E\|Z)$$

Neste contexto, o β é aplicado para distribuir o peso que cada um dos termos, a divergência de Kullback-Leibler (L_{rec}) e o erro de reconstrução ($D_{\text{KL}}(E\|Z)$), têm na função de perda:

$$\mathcal{L} = \beta L_{\text{rec}} + (1-\beta) D_{\text{KL}}(E\|Z)$$

Em termos simples, o beta permite um ajuste de ênfase entre a regularização do espaço latente (proporcionado pela KLD) e a precisão da reconstrução dos dados originais (proporcinado pelo erro de reconstrução). Por exemplo, considerando um beta com valor 0,7, estamos atribuindo 70% de relevância para o erro de reconstrução e 30% de relevância para a KLD. Os seguintes limiares podem ser interpretados:

- $\beta < 0{,}5$: a função de perda considera mais peso para a KLD, ou seja, a penalização pelo erro de reconstrução é diminuida. Isso pode resultar em um espaço latente mais regularizado, o que melhora a generalização e a interpolação. No entanto, isso pode sacrificar a precisão da reconstrução;
- $\beta = 0{,}5$: os dois termos são considerados igualmente;
- $\beta > 0{,}5$: a função de perda considera mais peso para o erro de reconstrução, ou

seja, a penalização pela divergência KL é reduzida. Isso favorece a precisão da reconstrução, mas pode resultar em um espaço latente menos organizado e mais propenso a overfitting.

O VAE que introduz o beta em sua função de erro recebe o nome “beta-VAE”. Quando comparados com os VAEs, o principal diferencial do beta-VAE é a possibilidade de exploração de diferentes vantagens entre a fidelidade da reconstrução e a regularização do espaço latente. Os resultados da aplicação da normalização dos dados de entrada serão abordados na Seção 4.4.

3. METODOLOGIA

O objetivo principal deste trabalho é fazer uma prova de conceito do uso de VAEs para gerar distribuições do momento transversal dos píons e dos káons, as partículas filhas do méson D^0. Esse será o primeiro passo de um objetivo maior: construir uma arquitetura que consiga gerar mésons D^0 de maneira massiva. Esse méson, por sua vez, pode ser obtido através da geração das 4 grandezas do seu quadrimomento, conforme explicado na Seção 2.1.1. A Fig. 3.1 resume as etapas necessárias para a reconstrução em massa deste méson.

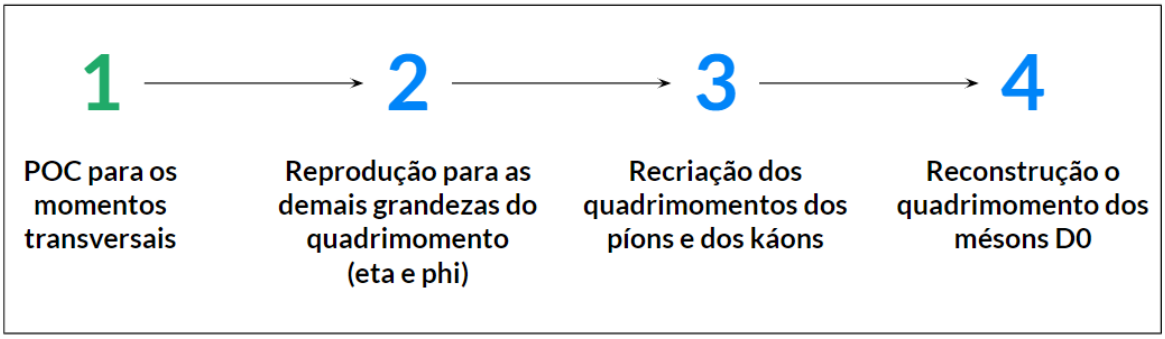

Figura 3.1: Passo a passo para o objetivo final de geração dos mésons D^0. O escopo desse projeto é a primeira etapa.

- **Etapa 1:** prova de conceito para gerar dados de uma das grandezas do quadrimomento do méson D^0, o momento transversal, através do VAE. Esse estudo será realizado em cima de dados simulados e é o escopo deste projeto.
- **Etapa 2:** tendo uma arquitetura de VAE que funciona para o objetivo da etapa 1, espera-se que essa rede neural também possa gerar dados coerentes das demais grandezas do quadrimomento: η e ϕ. Essa seria uma etapa futura, fora do escopo deste projeto.

- **Etapa 3:** com todas as componentes necessárias advindas dos VAEs, pode-se reconstruir o quadrimomento completo do píon e do káon. Essa seria uma etapa futura, fora do escopo deste projeto.
- **Etapa 4:** a partir dos quadrimomentos dos píons e káons, finalmente, pode-se reconstruir os mésons D^0. Essa seria uma etapa futura, fora do escopo deste projeto.

Obtendo sucesso nessas 4 etapas, terá-se criado uma máquina geradora de distribuições cinemáticas de mésons D^0. Dessa forma, as simulações de Monte Carlo podem se tornar desnecessárias para o objetivo de geração dessas distribuições, já que a estrutura estará preparada para usar como input dados reais das colisões.

3.1. CONJUNTO DE DADOS DE INPUT

Os dados de input são provenientes de simulações de colisões ultrarrelativísticas entre íons de chumbo no LHC com produto de colisões coletadas pelo detector CMS. Eles representam distribuições de momento transversal das partículas filhas dos mésons D^0 gerados nessas colisões. Essa grandeza define a quantidade de movimento da partícula na direção perpendicular ao feixe colidido, e é importante para determinar a energia e a direção das partículas na colisão (a Seção 2.1.1 contém mais informações sobre o momento transversal). O formato original do arquivo é uma TTree do Root [15] (software mantido pelo CERN). Esses dados são transformados em uma tabela salva em um arquivo no formato .txt e depois, dentro do programa que aplica o VAE, serão transformados em um tensor do PyTorch [16].

Na Fig. 3.2, apresentam-se as distribuições de momento transversal de píons e káons carregados resultantes do decaimento de mésons D^0 produzidos em simulações de Monte Carlo de colisões PbPb no LHC a uma energia no centro-de-massa por par de núcleons de 5,02 TeV. Como o detector CMS não consegue identificar os píons e káons, colocamos as distribuições dos produtos de decaimentos misturando os píons e os káons. Esse será o conjunto de dados utilizado neste projeto.

3.2. ESTRUTURA PADRÃO DO VAE

Os autoencoders variacionais possuem uma arquitetura-base padrão, que executa todas as funções apresentadas nas Seções 2.3 e 2.4. É evidente que, conforme os

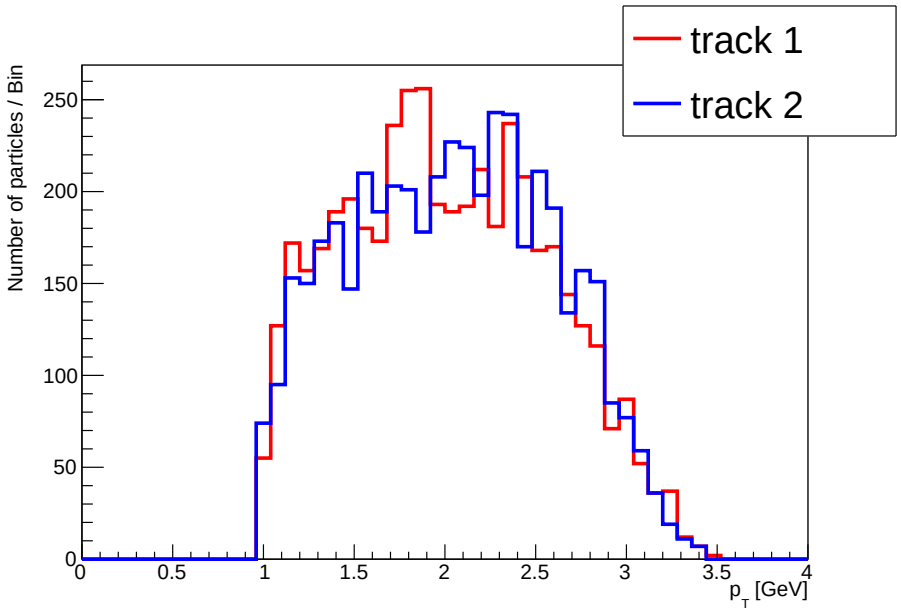

Figura 3.2: Distribuições de momento transversal dos produtos de decaimento dos mésons D^0 em simulações de Monte Carlo de colisões PbPb a energias do LHC. “track 1” e “track 2” representam as filhas com rótulo 1 e 2, respectivamente, ou seja, ambas as distribuições misturam as informações dos píons e dos káons.

resultados obtidos, serão necessários ajustes e adições à arquitetura padrão, conforme será abordado ao longo deste documento, porém, ela é o ponto de partida na criação de um VAE. Seguem os componentes de uma arquitetura padrão de autoencoder variacional:

- **Importar as bibliotecas necessárias**: a maioria está relacionada com o PyTorch, visto que a sua sintaxe limpa (e baseada em Python) facilita a implementação de arquiteturas de redes neurais complexas, como as necessárias para VAEs, sem sacrificar a clareza do código. Além disso, o PyTorch oferece módulos e funcionalidades específicas para redes neurais, incluindo otimizadores e funções de ativação, simplificando ainda mais a implementação de modelos.

- **Input/Declaração dos dados**: importação dos dados de entrada do VAE. Para esse projeto, isso ocorrerá conforme descrito na Seção 3.1.

- **Organização e aleatorização dos dados**: utilizar os dataloaders para organizar as amostras em listas aleatorizadas que serão utilizadas no treinamento da rede. Isso é essencial para promover a generalização do modelo, pois expõe a rede a diferentes contextos e variações nos dados. Tipicamente, os dados de entrada

são divididos em 3 sets: teste, treino e validação. Dessa forma, o modelo é exposto a uma variedade representativa de contextos, diminuindo as chances de overfitting e melhorando a convergência.

- **Criação da rede neural VAE**: construção do encoder, definição do tamanho da representação reduzida (dimensão do vetor latente) e declaração do decoder. Além disso, nessa etapa são definidos os valores dos hiperparâmetros do VAE.
- **Estrutura do otimizador**: aqui, será definido o otimizador, que irá guiar o processo de ajuste dos parâmetros do modelo durante o treinamento. Além disso, ele também será responsável por minimizar a função de erro durante o treinamento.
- **Estrutura de treinamento**: definição do processo de treinamento do VAE, utilizando um loop de epochs (processo em que todos os dados passam pela rede uma vez) para a execução de todos os batches (grupos de processamento dos dados). Isso irá ajustar os parâmetros do VAE para otimizar uma função de perda que compreende o equilíbrio entre a precisão na reconstrução dos dados e a coerência na distribuição latente.
- **Output**: geração de novos dados que seguem a distribuição do input.

Abaixo, segue um exemplo de um VAE com a arquitetura acima. Nesse caso, o input é uma distribuição normal com média 5 e desvio padrão 2. Neste exemplo, podemos perceber que a rede neural é boa em reproduzir os dados de treino e de teste, conforme as Figs. 3.3 e 3.4. Nelas, podemos observar as linhas azuis (reprodução do VAE), reproduzindo os datasets de treino e de teste (linhas verde e laranja, respectivamente) razoavelmente bem, com uma média visualmente próxima de 5. Porém, o VAE não obteve a mesma qualidade para gerar novos dados que seguem a mesma distribuição, como observado na Fig. 3.5, onde a reprodução do VAE (vermelho) se distancia da distribuição normal definida (azul).

Existem diversas maneiras de melhorar esse resultado e a performance do VAE para a geração de novos dados. Este será o esforço principal do projeto e está elaborado no Capítulo 4.

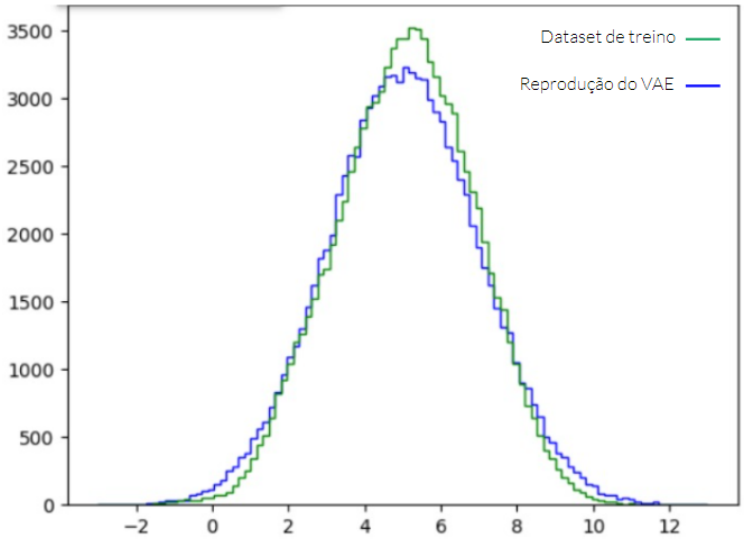

Figura 3.3: Gráficos ilustrando a execução do VAE buscando reproduzir o dataset de treino.

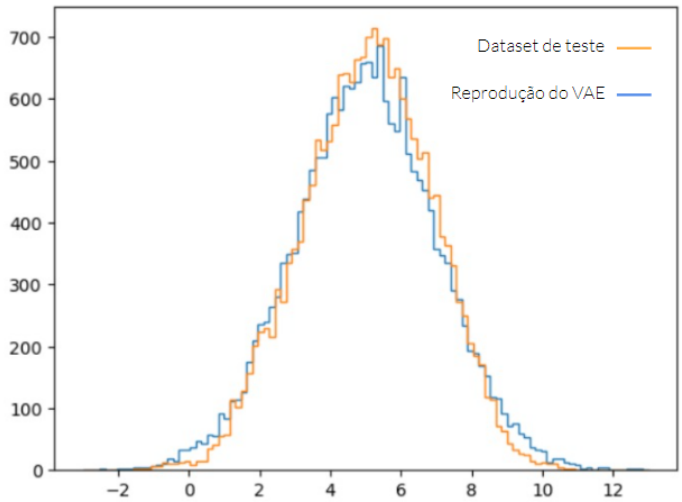

Figura 3.4: Gráficos ilustrando a execução do VAE buscando reproduzir o dataset de teste.

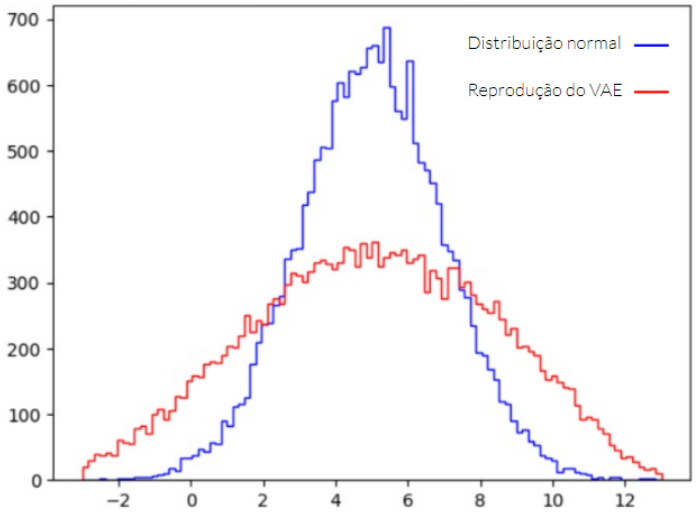

Figura 3.5: Gráficos ilustrando a execução do VAE buscando gerar novos dados que sigam a distribuição declarada.

3.3. TESTE KOLMOGOROV-SMIRNOV

Antes da construção do VAE, precisa-se definir uma métrica de sucesso quantitativa para o resultado desejado. Dessa forma, existirá uma maneira de entender quanto cada ajuste e implementação do VAE está se aproximando do sucesso. O teste de Kolmogorov-Smirnov (KS) é um método estatístico utilizado para comparar distribuições de dados. Em essência, ele mede a diferença entre as distribuições de duas amostras e verifica se essas diferenças são estatisticamente significativas. Ele é, portanto, muito útil para esse projeto.

O teste KS fornece uma ferramenta quantitativa para mensurar a distância entre as distribuições de entrada (dados originais) e as distribuições de saída (dados reconstruídos e gerados pelo VAE). Esta métrica é essencial para avaliar a performance do VAE, permitindo identificar se o modelo está conseguindo capturar a estrutura dos dados de entrada. Uma pequena distância entre as distribuições sugere que o VAE está aprendendo bem a representação dos dados, enquanto uma grande distância indica que o modelo está falhando em capturar a essência dos dados originais.

O teste KS compara distribuições de dados e retorna duas métricas: o valor estatístico (D) e o p-valor. Para obter esses valores, o teste KS calcula a maior diferença absoluta entre as funções de distribuição cumulativa (CDFs) das

distribuições comparadas. A CDF é uma função que, para qualquer valor dado, fornece a proporção de elementos da amostra que são menores ou iguais a esse valor. Segue um exemplo de funcionamento do teste KS, com duas amostras de dados supostas:

- Amostra A: [1, 2, 3, 4, 5]
- Amostra B: [1, 2, 2, 3, 3, 4, 5]

Para calcular a CDF da Amostra A, organiza-se os dados em ordem crescente e calcula-se a proporção cumulativa:

- Para o valor 1: CDF(1) = 1/5 = 0,2
- Para o valor 2: CDF(2) = 2/5 = 0,4
- Para o valor 3: CDF(3) = 3/5 = 0,6
- Para o valor 4: CDF(4) = 4/5 = 0,8
- Para o valor 5: CDF(5) = 5/5 = 1,0

Para a Amostra B, calcula-se a CDF da mesma forma:

- Para o valor 1: CDF(1) = 1/7 $\approx$ 0,14
- Para o valor 2: CDF(2) = 3/7 $\approx$ 0,43
- Para o valor 3: CDF(3) = 5/7 $\approx$ 0,71
- Para o valor 4: CDF(4) = 6/7 $\approx$ 0,86
- Para o valor 5: CDF(5) = 7/7 = 1,0

A estatística D é a maior diferença absoluta entre as CDFs das duas amostras para qualquer valor específico. Essas diferenças são:

- Para o valor 1: |CDF_A(1) - CDF_B(1)| = |0,2 - 0,14| = 0,06
- Para o valor 2: |CDF_A(2) - CDF_B(2)| = |0,4 - 0,43| = 0,03
- Para o valor 3: |CDF_A(3) - CDF_B(3)| = |0,6 - 0,71| = 0,11
- Para o valor 4: |CDF_A(4) - CDF_B(4)| = |0,8 - 0,86| = 0,06

- Para o valor 5: |CDF_A(5) - CDF_B(5)| = |1,0 - 1,0| = 0,0

A maior diferença absoluta entre as distribuições, que é o valor de D, é 0,11. Valores menores de D indicam que as duas distribuições são mais similares.

O p-valor, por sua vez, baseia-se em uma ”hipótese nula” de que as amostras de dados vêm da mesma distribuição. Ele fornece uma medida de quão provável é obter uma diferença tão grande (ou maior) que D entre as amostras, se elas pertencem à mesma distribuição. Por exemplo, se o p-valor resultante neste caso fosse 0,8, isso indicaria que há uma probabilidade de 80% de observar uma diferença de 0,11 (ou maior) entre as CDFs, assumindo que as amostras provêm da mesma distribuição. Portanto, não se rejeita a hipótese nula e conclui-se que as distribuições podem ser muito semelhantes.

O p-valor é calculado com base na estatística D, de maneira inversamente proporcional a ela. Esse cálculo utiliza uma fórmula derivada da função de distribuição cumulativa do KS, que não é simples de escrever sem recursos matemáticos muito avançados e que, portanto, não serão aprofundados neste projeto. Para ler mais sobre o D e o p-valor, acessar a referência [17].

Para este trabalho, o importante é entender que o p-valor determina se a diferença observada entre as duas distribuições é estatisticamente significativa. Tipicamente, os seguintes limiares são utilizados (também descritos em [17]):

- $p \leq 0{,}05$: Indica que a diferença entre as distribuições é estatisticamente significativa. A hipótese nula de que as distribuições são iguais é rejeitada. Para o VAE deste projeto, um p-valor igual ou inferior a 0,05 indica falha, pois as distribuições ainda não são suficientemente próximas.

- $p > 0{,}05$: Indica que não há diferença estatisticamente significativa entre as distribuições comparadas. Não há evidência suficiente para rejeitar a hipótese nula. Nesse caso, aceita-se que as distribuições são muito semelhantes. Para o VAE deste projeto, um p-valor maior que 0,05 indica sucesso.

- Para p-valores ainda mais altos, como 0,3 ou 0,4, a evidência é extremamente forte de que não há diferença significativa entre as distribuições.

No contexto deste projeto, o teste KS é aplicado para comparar a distribuição dos

dados de entrada com a distribuição dos dados reconstruídos pelo VAE e a distribuição dos dados gerados pelo VAE.

O VAE é projetado para aprender a representar os dados de entrada em um espaço latente de forma que a reconstrução dos dados a partir desse espaço seja o mais fiel possível aos dados originais. Ao aplicar o teste KS, pode-se quantitativamente avaliar o quanto a distribuição dos dados reconstruídos e dos dados gerados se assemelha à distribuição dos dados originais. Isso nos fornece uma medida objetiva da performance do VAE. Portanto, a partir das métricas do teste KS, o objetivo do projeto é obter um VAE com valores baixos para a estatística D, e altos para o p-valor (mirando em 0,05).

4. RESULTADOS

A partir da estrutura padrão do VAE, e de todas as suas ferramentas adicionais descritas na Seção 2.5, pode-se iniciar as tentativas de geração de dados com a mesma distribuição dos dados de entrada (momentos transversais dos píons e dos káons). Para cada modelo e iteração de VAE executada, o objetivo é que o output da rede neural replique a distribuição da Fig. 4.1, e a métrica de sucesso será o p-valor, com meta de 0,05.

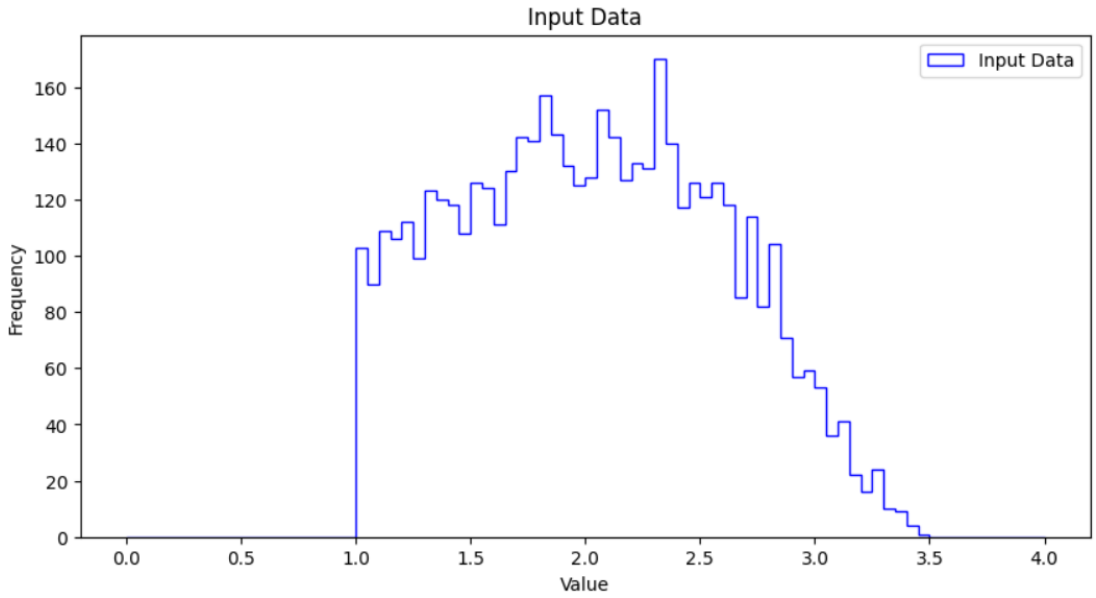

Figura 4.1: Distribuição dos dados de entrada. O objetivo do projeto é criar um VAE que possa gerar dados seguindo essa mesma distribuição.

A cada seção desse capítulo será adicionada uma nova ferramenta à estrutura padrão do VAE, permitindo que os resultados sejam progressivamente melhores.

4.1. CONSTRUÇÃO BÁSICA DO VAE

Com a estrutura padrão do VAE, descrita na Seção 3.2, e uma métrica de sucesso, descrita na Seção 3.2.1, pode-se construir a primeira versão do autoencoder variacional do projeto. A Fig. 4.2 demonstra, através da linha azul, a distribuição dos dados de entrada, enquanto a linha vermelha representa a geração de novos dados do primeiro VAE do projeto. Pode-se rapidamente perceber que esse primeiro resultado está ruim, visto que as duas distribuições são muito diferentes.

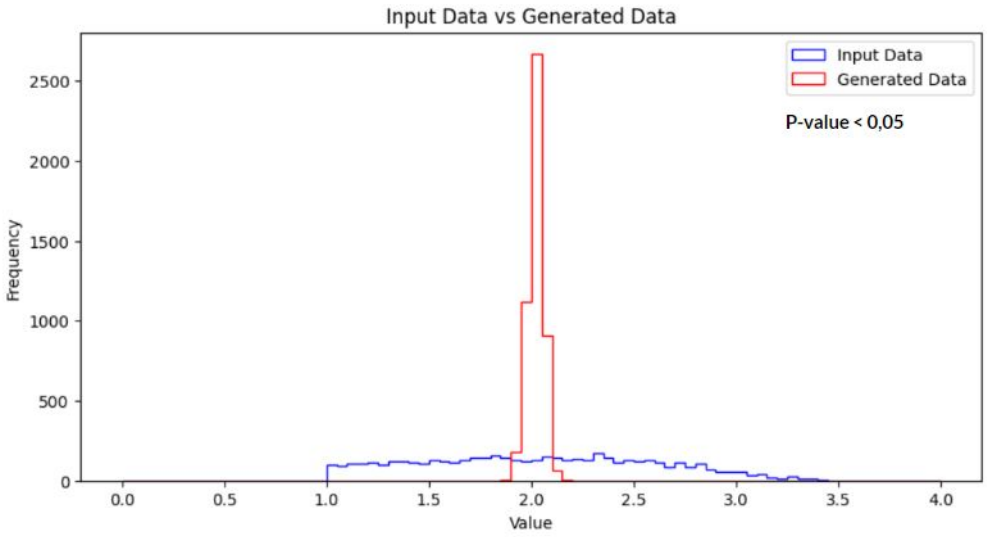

Figura 4.2: Geração de novos dados através da primeira arquitetura de VAE do projeto (estrutura padrão). Pode-se perceber que os novos dados gerados (linha vermelha) estão muito diferentes dos dados originais (linha azul).

Isso ocorre porque os hiperparâmetros da rede neural ainda não foram ajustados para otimizar a performance do VAE. Por ora, apenas os valores-padrão, presentes na Fig. 4.3, foram utilizados. Conforme comentado na Seção 2.4.4, é esperado que o resultado do VAE sem ajustes nos hiperparâmetros seja ruim, pois eles necessitam de adaptações para entender e replicar o comportamento dos dados de entrada. Neste resultado, o p-valor foi menor 0,05, indicando necessidade de melhoria.

Dessa forma, pode-se concluir que a geração de novos dados do VAE na Fig. 4.2 está ruim porque o VAE foi construído com valores padrão para os hiperparâmetros, sem otimizações. Eles necessitam de ajustes e estão distantes do seu valor ideal.

Encontrar o melhor valor para um hiperparâmetro é uma tarefa difícil e demorada,

Hiperparâmetro	Valor-padrão utilizado
Número de camadas no VAE	2
Dimensão de cada camada	128
Dimensão do vetor latente	8
Tamanho do batch	32
Taxa de aprendizado	2E-3
Função de ativação utilizada	tanh
Otimizador utilizado	Adam

Figura 4.3: Valores-padrão para os 7 hiperparâmetros do VAE. Eles foram utilizados para a primeira versão do VAE do projeto.

principalmente se for feita manualmente. Cada hiperparâmetro pode ter um impacto significativo no desempenho do modelo, e encontrar a combinação ideal entre eles geralmente requer testar muitas configurações diferentes.

A Fig. 4.4 mostra o resultado do mesmo VAE, porém com uma mudança em um de seus hiperparâmetros: o tamanho das camadas ocultas foi alterado para 64. Mesmo com apenas uma alteração dentre todos os hiperparâmetros, pode-se notar uma diferença muito significativa no output.

4.2. INSERÇÃO DO OPTUNA

A partir dos resultados insatisfatórios com a arquitetura padrão do VAE, essa seção foca em demonstrar os resultados ao inserir o otimizador de hiperparâmetros Optuna. Conforme descrito na Seção 2.5.1, precisa-se “comunicar” ao VAE um range de opções coerentes para cada hiperparâmetro, bem como o número de trials desejados, e o trabalho do Optuna será achar os valores com o menor erro de validação nesse intervalo através dos trials. Para o VAE desse projeto, os limiares escolhidos para cada

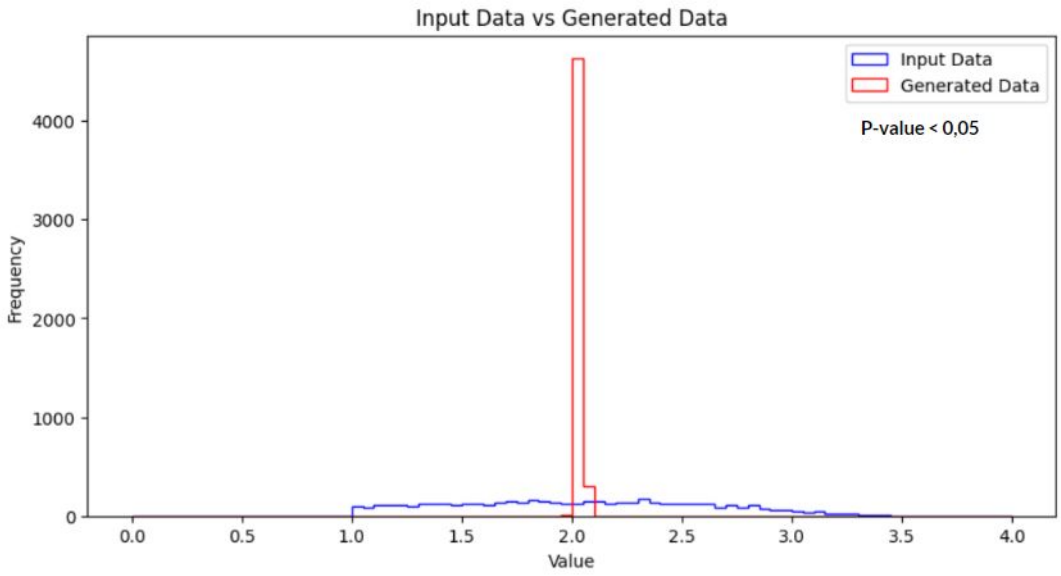

Figura 4.4: Geração de novos dados utilizando primeira arquitetura de VAE do projeto, porém com o tamanho das camadas ocultas como 64.

hiperparâmetro são apresentados nas Fig. 4.5 e 4.6. O número de trials foi de 25.

A Fig. 4.7 demonstra o output do VAE com a instalação do Optuna na arquitetura da rede neural. É notável que a performance do VAE melhorou com relação ao resultado utilizando apenas a arquitetura padrão. A distribuição dos dados gerados ficou mais "larga", se ajustando melhor à linha azul. Para atingir esse resultado, os valores encontrados pelo Optuna para cada hiperparâmetro estão descritos na Fig. 4.8.

Dessa forma, pode-se interpretar que o Optuna elevou a qualidade do VAE na aplicação do projeto, já que as distribuições estão mais similares. Mesmo assim, o p-valor superior a 0,05 ainda não foi obtido.

Analisando o desempenho do treinamento do VAE, pode-se entender que o seu output ainda está abaixo do esperado porque o seu processo de aprendizagem está instável e ineficiente. A Fig. 4.9 demonstra o processo final de treinamento com os melhores hiperparâmetros elegidos pelo Optuna. Na linha azul, tem-se o cálculo da perda no dataset de treino, enquanto a linha laranja mostra o cálculo da perda no dataset de validação. Cada intervalo no eixo horizontal representa uma epoch do treinamento. O comportamento esperado, com um treinamento estável, é a diminuição constante

Hiperparâmetro	Valor mínimo	Valor máximo
Número de camadas no VAE	2	5
Dimensão de cada camada	32	256
Dimensão do vetor latente	2	32
Tamanho do batch	8	64
Taxa de aprendizado	1E-5	1E-2

Figura 4.5: Intervalo de valores para os hiperparâmetros comunicados ao Optuna dentro da função objetivo.

Hiperparâmetro	Opção 1	Opção 2	Opção 3
Função de ativação utilizada	relu	tanh	leaky_relu
Otimizador utilizado	Adam	RMSprop	-

Figura 4.6: Opções de função de ativação e otimizador comunicados ao Optuna dentro da função objetivo.

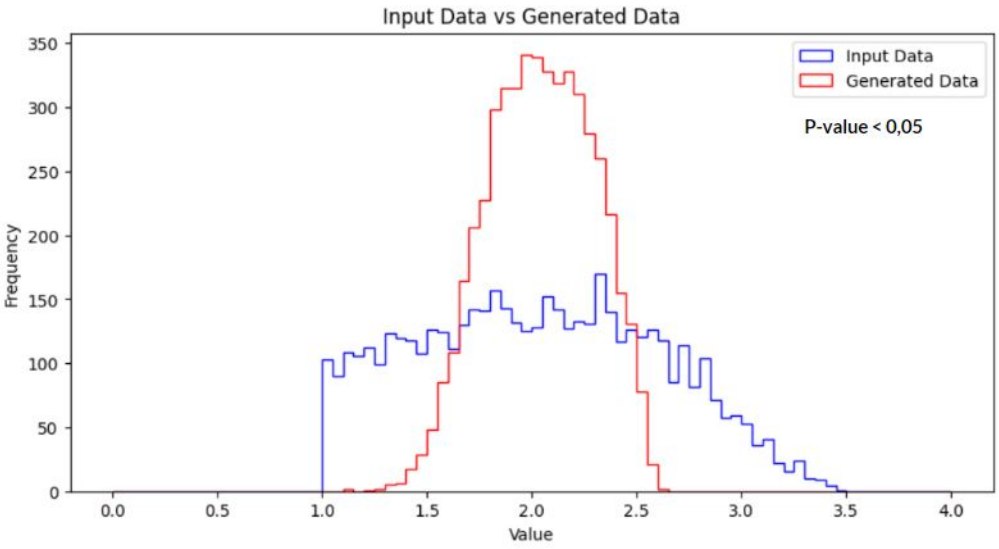

Figura 4.7: Geração de dados do VAE com a inserção do Optuna.

Hiperparâmetros	Valor otimizado pelo Optuna
Número de camadas no VAE	3
Dimensão de cada camada	88, 69 e 118
Dimensão do vetor latente	28
Tamanho do batch	12
Taxa de aprendizado	7,9E-4
Função de ativação utilizada	tanh
Otimizador utilizado	Adam

Figura 4.8: Valores otimizados dos hiperparâmetros através do Optuna.

de ambas as linhas, com um aprendizado que garante um erro sempre menor. Porém, esta imagem mostra que as linhas de perda estão flutuando significativamente e que estão com pouca diminuição do erro (sem diminuição para valores abaixo de 0,6, por exemplo), evidenciando que o treinamento está instável, prejudicando o aprendizado do modelo. O valor do erro de validação na última epoch, por exemplo, é muito maior que diversos valores anteriores.

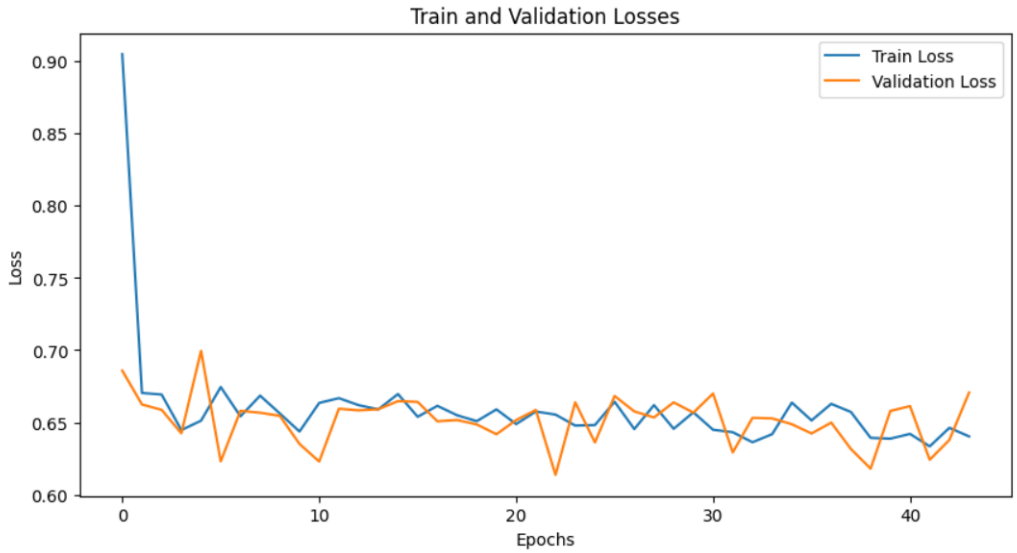

Figura 4.9: Erros de teste e de validação durante o treinamento do VAE. Pode-se notar que os valores flutuam muito e que o erro não está diminuindo, pois o treinamento está instável.

Dessa forma, embora o Optuna tenha sido útil, ele não resolveu completamente os problemas da rede neural, visto que o processo de treinamento do VAE está instável. Isso exige a introdução de outra ferramenta que possa ajudar a estabilizar o treinamento do modelo.

4.3. INSERÇÃO DA NORMALIZAÇÃO DOS DADOS DE ENTRADA

No contexto do VAE, a normalização dos dados de entrada deve ser integrada no início do código. Antes de alimentar os dados ao modelo, eles são normalizados. Após o modelo gerar novos dados, esses são desnormalizados para a escala original, garantindo que os resultados sejam interpretáveis no mesmo contexto dos dados de entrada. Um breve resumo do fluxo que será implementado pode ser visto na Fig. 4.10.

Antes de analisarmos os resultados do VAE com essa novidade, é válido relembrar

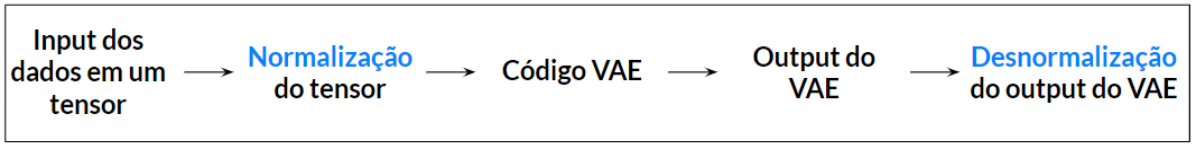

Figura 4.10: Resumo da adição da normalização e, por consequência, da desnormalização, no fluxo do VAE.

que, além da normalização dos dados, o código executado é exatamente o mesmo da seção anterior, ou seja, ainda conta com o otimizador de hiperparâmetros Optuna. A Fig. 4.11 demonstra o output normalizado dessa versão do VAE, e pode-se notar que os impactos de inserir a normalização de dados são muito positivos. É evidente que as linhas vermelha e azul estão muito mais próximas que nas interações anteriores. A altura e a largura de ambas as distribuições têm, visualmente, a mesma ordem de grandeza.

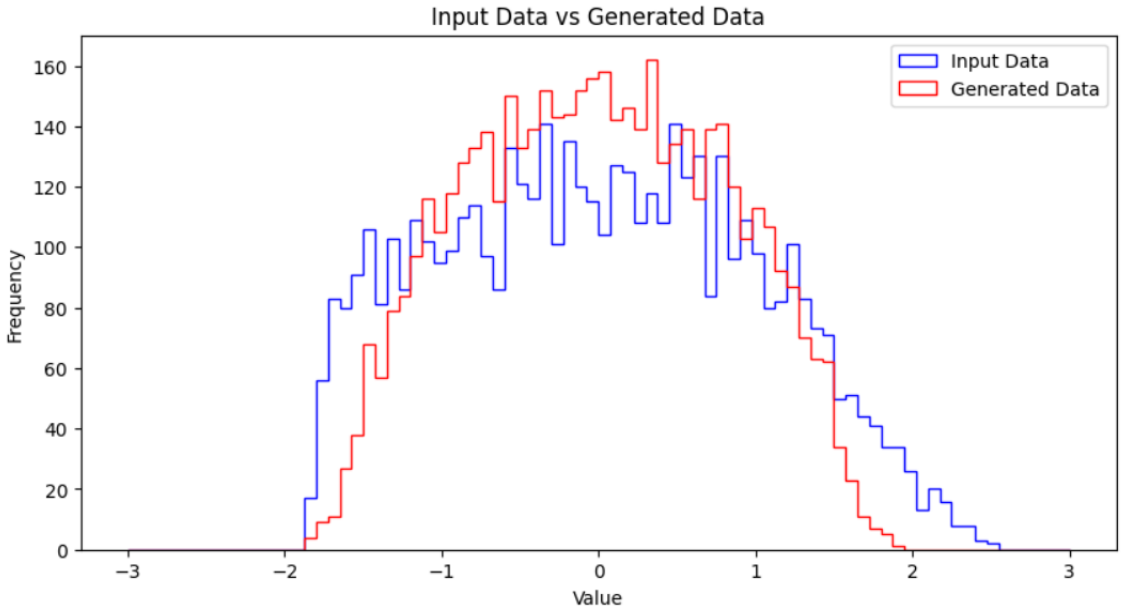

Figura 4.11: Geração de novos dados (vermelho) normalizados, após a inclusão da ferramenta de normalização.

Assim como introduzido na Fig. 4.10, ainda deve-se desnormalizar esse resultado. O output desnormalizado está na Fig. 4.12. Pode-se notar que a performance está muito semelhante à versão normalizada, com um desempenho muito melhor que anteriormente. A distribuição dos dados gerados e dos dados de entrada estão muito mais semelhantes com a normalização dos dados, que se provou uma etapa crucial e

indispensável para o VAE do projeto.

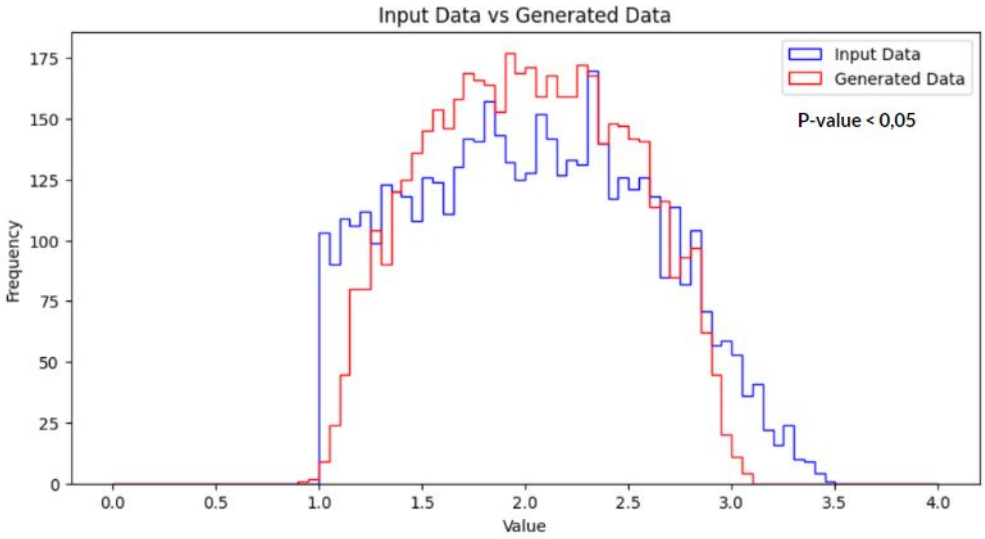

Figura 4.12: Geração de novos dados (vermelho) desnormalizados, após a inclusão da ferramenta de normalização.

A principal causa dessa melhora no resultado após a normalização dos dados de entrada é que, conforme introduzido na Seção 2.5.2, o processo de treinamento agora está muito mais estável e eficiente. A Fig. 4.13 mostra a diminuição do erro de treino e de validação durante o treinamento final do VAE. Dessa vez, os valores estão diminuindo consistentemente, com poucas flutuações ou instabilidades. Os valores de erro estão agora em torno de 0,2, enquanto flutuavam perto de 0,65 na Fig 4.9. Isso demonstra que a aprendizagem da rede neural está muito melhor agora, habilitando um output muito aproximado dos dados de entrada.

Mesmo que os resultados estejam muito melhores, o VAE ainda não atingiu a métrica de sucesso, p-valor $> 0{,}05$. Para isso, novamente, é necessário introduzir novas ferramentas que podem aumentar a qualidade do output.

4.4. INSERÇÃO DO BETA

Mesmo com as melhorias apresentadas anteriormente, a métrica de sucesso do projeto ainda não foi atingida. Dessa forma, a ferramenta descrita na Seção 2.5.3 será implementada. A introdução do beta na função de perda do VAE proporciona diferentes relevâncias entre os termos da função de perda. Achar o melhor valor para o beta não é uma tarefa trivial, porém, tentativas manuais podem apontar qual é a direção correta.

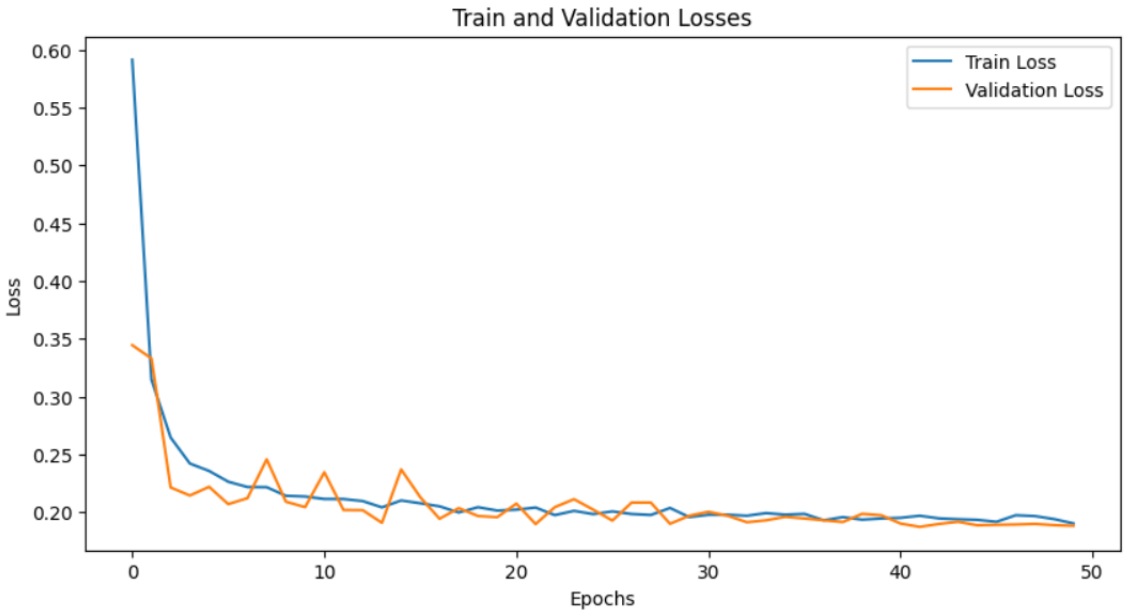

Figura 4.13: Erros de teste e de validação durante o treinamento do VAE pós normalização dos dados de entrada. Pode-se notar uma flutuação muito menor, esse treinamento está estável.

Abaixo, seguem os outputs do VAE com diferentes valores de β. A Fig. 4.14 mostra a geração de novos dados com $\beta = 0{,}1$. Esse valor de beta significa que o VAE está dando um peso muito maior à KLD do que ao erro de reconstrução durante o treinamento. Nessa imagem, pode-se perceber que a performance foi muito ruim - a distribuição dos dados gerados está extremante distante da distribuição dos dados de entrada. Nesse caso, o p-valor ainda é inferior a 0,05.

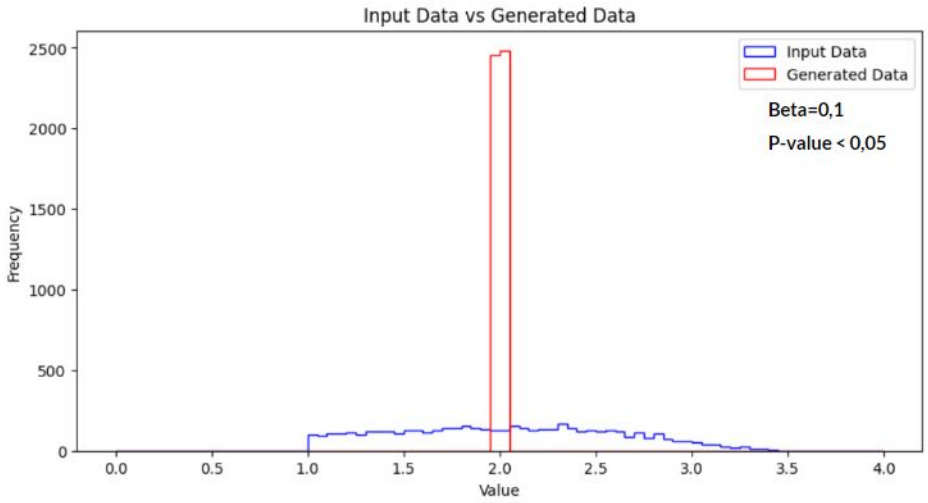

Figura 4.14: Geração de novos dados (vermelho) com $\beta = 0{,}1$.

Seguindo com o testes manuais, a Fig. 4.15 mostra a geração de novos dados com $\beta = 0{,}3$. A contribuição do erro de reconstrução aumentou, porém ainda é menor que a da KLD durante o treinamento. Os resultados melhoraram consideravelmente, porém o p-valor ainda não atingiu a métrica definida.

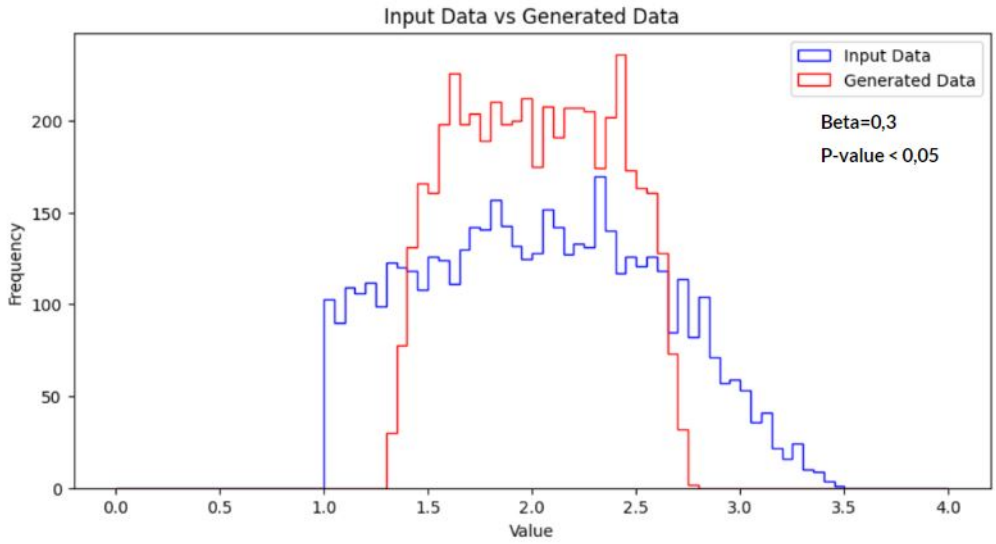

Figura 4.15: Geração de novos dados (vermelho) com $\beta = 0{,}3$.

Aumentar o valor de beta parece ter melhorado o output. A Fig. 4.16 mostra a performance do VAE com $\beta = 0{,}5$. Novamente, percebe-se melhora na geração de dados (os gráficos estão mais similares).

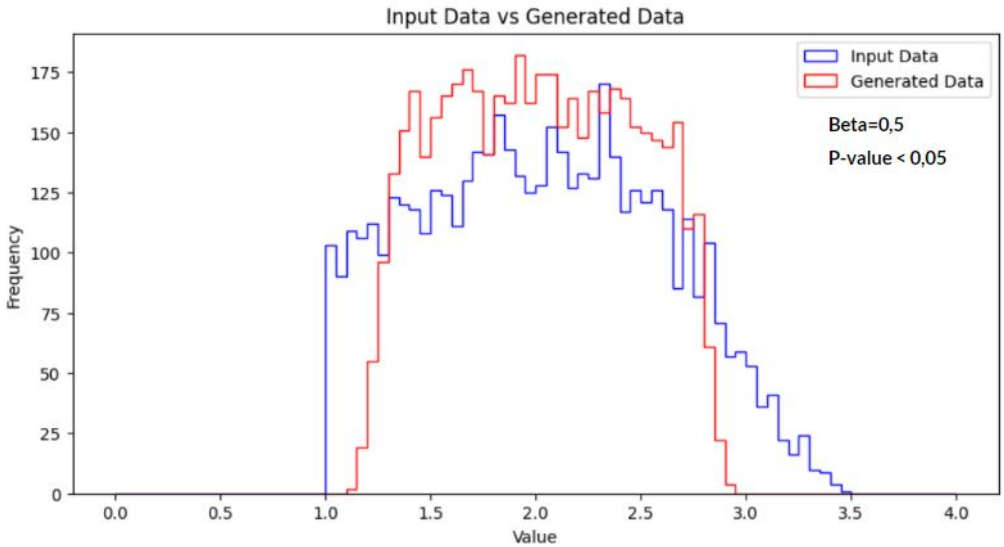

Figura 4.16: Geração de novos dados (vermelho) com $\beta = 0{,}5$.

Finalmente, na Fig. 4.17, tem-se o output do VAE com uma relevância maior do erro de reconstrução com relação à KLD. Com $\beta = 0{,}7$, a geração de novos dados está com uma distribuição visualmente parecida com os dados de entrada, porém com um p-valor ainda inferior a 0,05.

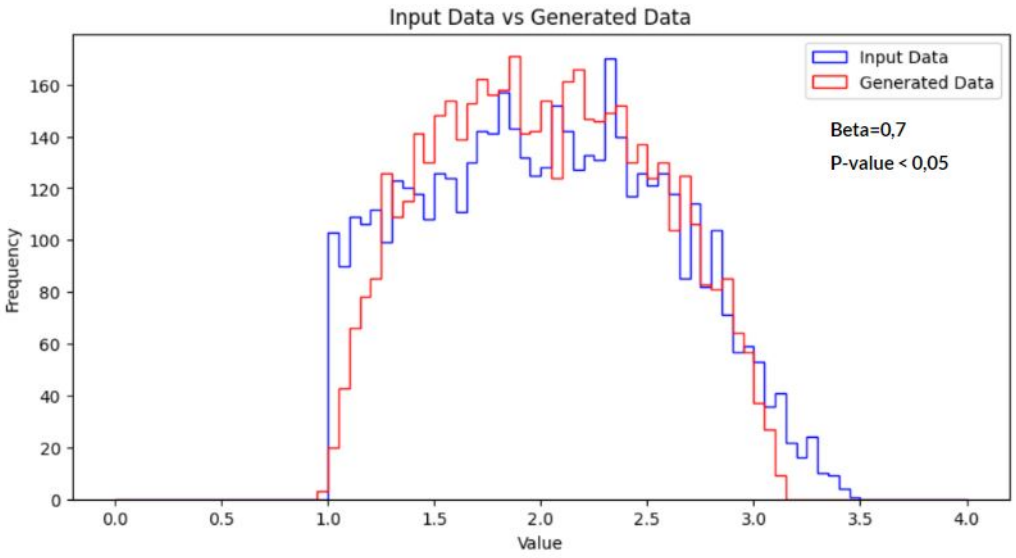

Figura 4.17: Geração de novos dados (vermelho) com $\beta = 0{,}7$.

O aumento da relevância do erro de reconstrução está melhorando muito a performance do VAE. A Fig. 4.18 mostra o output do VAE com $\beta = 0{,}9$, dando 90% de contribuição para o termo do erro de reconstrução na função de perda. Novamente, percebe-se uma melhora na semelhança dos gráficos, ainda sem atingir 0,05 para o p-valor.

Utilizando $\beta = 0{,}95$, obtém-se o primeiro VAE a atingir a métrica de sucesso definida, com p-valor $> 0{,}05$. Esse p-valor indica que as duas distribuições são extremamente semelhantes, isto é, de acordo com a métrica definida, não existem evidências para rejeitar a hipótese nula de que as distribuições são iguais. A Fig. 4.19 mostra que, com essa distribuição de pesos na função de perda, a geração de novos dados do VAE é extremamente parecida com os dados de entrada.

Até aqui, a qualidade do output do VAE aumentou junto com o aumento de β. Porém, após o sucesso obtido com $\beta = 0{,}95$, o aumento do beta deixa de melhorar a performance do VAE. As Figs. 4.20 e 4.21 mostram a performance do VAE com $\beta = 0{,}98$

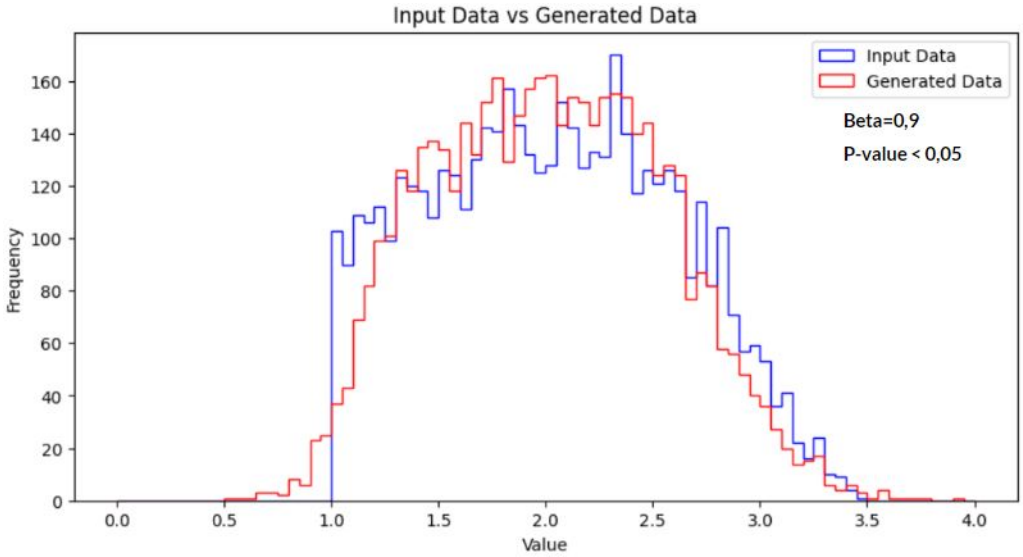

Figura 4.18: Geração de novos dados (vermelho) com $\beta = 0{,}9$.

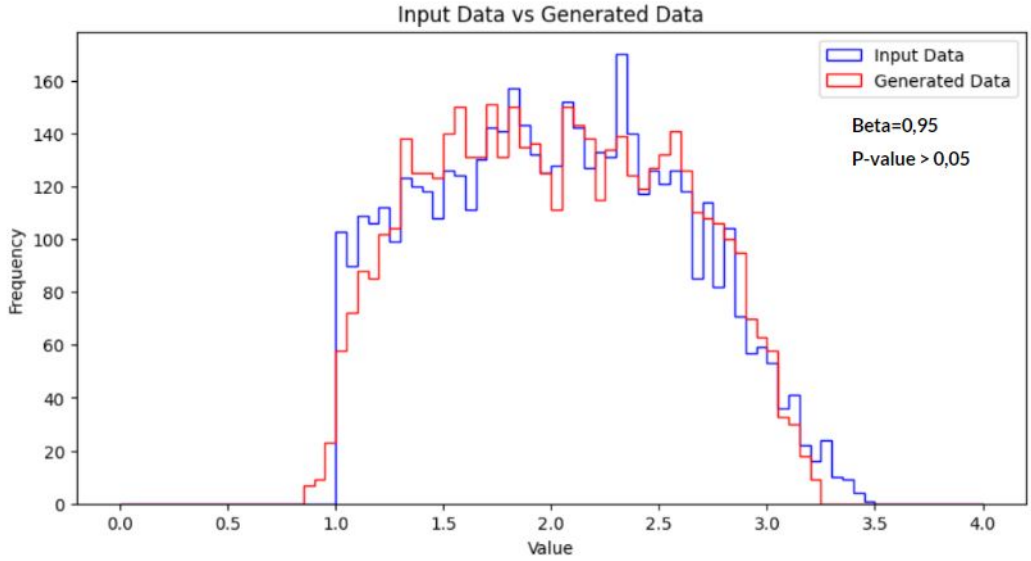

Figura 4.19: Geração de novos dados (vermelho) com $\beta = 0{,}95$.

e $0{,}99$, respectivamente. Através desses gráficos, pode-se perceber que, a partir de $0{,}95$, aproximar o beta de 1 pode gerar perda de performance.

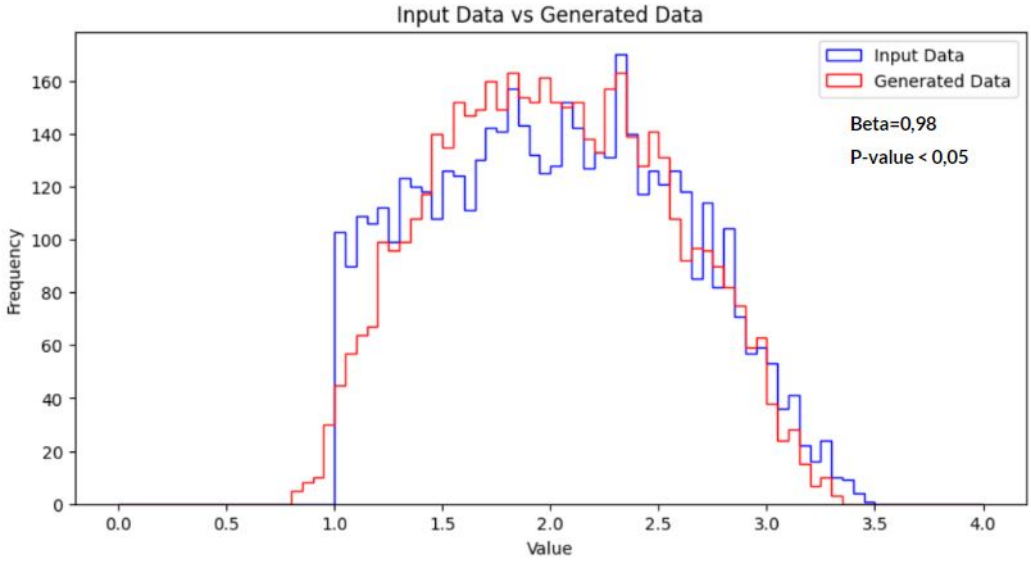

Figura 4.20: Geração de novos dados (vermelho) com $\beta = 0{,}98$.

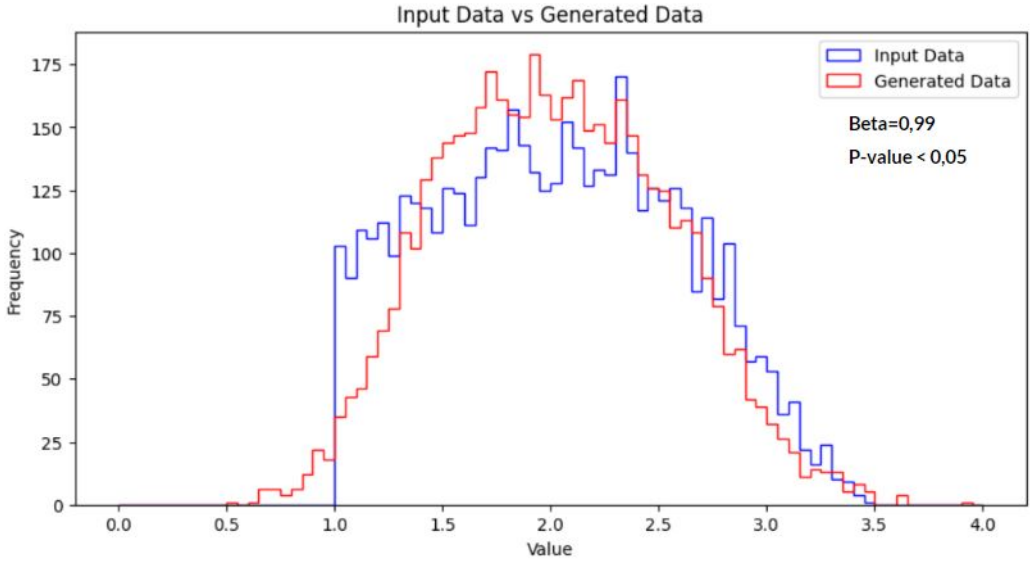

Figura 4.21: Geração de novos dados (vermelho) com $\beta = 0{,}99$.

5. CONCLUSÃO

No campo da física de partículas, a identificação e caracterização de diferentes mésons é de extrema importância para o estudo dos fenômenos subatômicos. As distribuições cinemáticas dos mésons D^0 podem ser obtidas utilizando simulações de Monte Carlo, um método que pode conter considerável taxa de ruído e baixa eficiência operacional, necessitando da ordem de meses para entregar os resultados necessários para um conjunto de dados da ordem de milhões de eventos.

Este trabalho teve como objetivo executar o primeiro passo para o levantamento de um método alternativo na obtenção de distribuições cinemáticas dos mésons D^0: a utilização de autoencoders variacionais. Este primeiro passo trata-se da utilização desta rede neural para a geração de distribuições do momento transversal dos píons e káons, partículas filhas provenientes do decaimento dos mésons D^0. Obtendo sucesso nessa missão, essa arquitetura poderá ser replicada no futuro para as outras grandezas do quadrimomento dos píons e dos káons, entregando, finalmente, a geração de mésons D^0. A motivação por trás deste estudo, portando, está na busca de um método mais eficiente e assertivo do que as simulações de Monte Carlo, já que o uso de VAEs pode reduzir significativamente o tempo e os recursos computacionais necessários.

Os resultados alcançados demonstraram uma evolução significativa desde a estrutura padrão do VAE, que inicialmente apresentou resultados insatisfatórios. A inclusão de ferramentas adicionais, como a biblioteca Optuna, a normalização dos dados de entrada, e a adaptação para beta-VAE, foram cruciais para a melhoria do desempenho. O Optuna ajudou a ajustar automaticamente os hiperparâmetros do modelo, resultando em uma melhor configuração sem a necessidade de tentativas manuais exaustivas. A normalização dos dados estabilizou o processo de treinamento, permitindo que o modelo aprendesse de forma mais eficiente. Finalmente, a introdução do beta-VAE permitiu controlar a contribuição da KLD e do erro de reconstrução na

função de perda, equilibrando melhor as vantagens entre a fidelidade da reconstrução e a regularização. Com essas melhorias, conseguiu-se alcançar um p-valor superior a 0,05, indicando que os dados gerados pelo VAE seguem uma distribuição estatisticamente muito semelhante à distribuição dos dados de entrada. Com a versão final do VAE apresentada no trabalho, pode-se afirmar que a prova de conceito para a geração da distribuição dos momentos transversais dos píons e dos káons foi um sucesso. A Fig. 5.1 retoma a progressão do p-valor de acordo com as melhorias agregadas à arquitetura padrão do VAE.

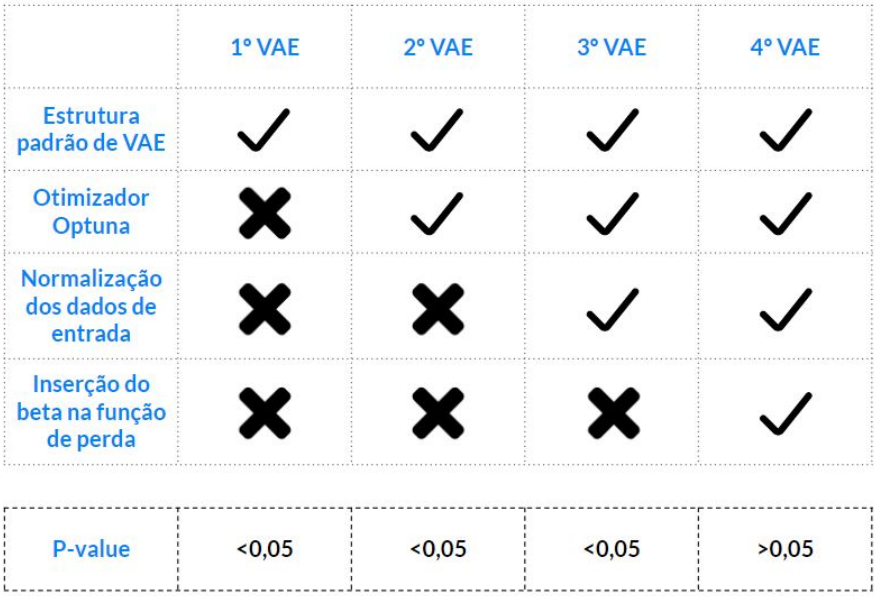

Figura 5.1: Resumo do progresso do p-valor a cada nova ferramenta adicionada ao modelo padrão do VAE.

Seguindo com o objetivo global de gerar mésons D^0, a próxima etapa do projeto envolve a geração das demais componentes do quadrimomento (pseudorapidez, η, e ângulo azimutal, ϕ) dos píons e káons, utilizando VAEs. A capacidade de gerar todas as componentes do quadrimomento permitirá a reconstrução completa dos quadrimomentos dessas partículas, e, eventualmente, dos mésons D^0. Além disso, a partir dos pontos testados e aprendidos com este projeto, existem oportunidades para futuros trabalhos que utilizem VAEs. Segue uma lista de itens que podem ser utilizados para garantir uma performance ainda melhor do VAE, que não foram exploradas neste projeto:

- **Grid search**: a grid search é uma técnica de otimização de hiperparâmetros que

envolve a avaliação exaustiva de todas as combinações possíveis de valores, buscando identificar a melhor configuração para a rede neural. Enquanto o Optuna decide um ponto de partida e otimiza a partir dele, o grid search foca em testar literalmente todas as permutações entre todos os possíveis valores de cada hiperparâmetro. Esse método não foi utilizado neste projeto pelo tempo de processamente requerido, porém ele pode ser explorado futuramente. Apesar do grande tempo de processamento, ele garante que a melhor combinação de hiperparâmetros será encontrada.

- **Otimizar o melhor valor exato de beta**: conforme descrito na Seção 4.4, os valores do parâmetro β foram testados manualmente neste projeto. Mesmo que tenha sido possível encontrar um valor aproximado do “melhor beta”, 0,95, utilizar uma ferramenta de otimização para esse parâmetro com certeza aumentaria ainda mais a qualidade de output do VAE.

- **Demais arquiteturas de VAE**: além do VAE convencional e do beta-VAE abordados no projeto, existem outras arquiteturas de VAE mais complexas, como os Hierarchical VAEs ou Conditional VAEs, que poderiam capturar ainda mais nuances nos dados e gerar um output mais preciso.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] The Large Hadron Collider. Disponível em: https://home.cern/science/accelerators/large-hadron-collider. Acesso em: 06/05/2024.

[2] Roman Pasechnik and Michal Šumbera. “ Phenomenological review on quark–gluon plasma: concepts vs. observations.” Universe, 3(1):7, 2017.

[3] Orzari, Breno and Chernyavskaya, Nadezda and Cobe, Raphael and Duarte, Javier and Fialho, Jefferson and Gunopulos, Dimitrios and Kansal, Raghav and Pierini, Maurizio and Tomei, Thiago and Touranakou, Mary,"LHC hadronic jet generation using convolutional variational autoencoders with normalizing flows", Mach. Learn. Sci. Tech., 4, 045023, 2023. 10.1088/2632-2153/ad04ea. Disponível em: https://iopscience.iop.org/article/10.1088/2632-2153/ad04ea.

[4] Diederik P. Kingma and Max Welling (2019), “An Introduction to Variational Autoencoders”, Foundations and Trends in Machine Learning: Vol. 12, No. 4, pp 307–392. DOI: 10.1561/2200000056.

[5] R. L. Workman et al. (Particle Data Group), “The Review of Particle Physics”, Prog. Theor. Exp. Phys. 2022, 083C01 (2022) and 2023 update.

[6] CERN Accelerating science (CMS). Disponível em: https://home.cern/science/experiments/cms. Acesso em 04/08/2024.

[7] Busza, Wit and Rajagopal, Krishna and van der Schee, Wilke, "Heavy Ion Collisions: The Big Picture, and the Big Questions", Ann. Rev. Nucl. Part. Sci., 68, 339–376, 2018. 10.1146/annurev-nucl-101917-020852.

[8] Ian Goodfellow and Yoshua Bengio and Aaron Courville, "Deep Learning - An MIT Press Book", published by MIT Press, 2016. Available at http://www.deeplearningbook.org.

[9] Dive into deep learning. Capítulo 4 - Perceptrons multicamada Parte 1.2.1 - Função ReLU - Disponível em: https://pt.d2l.ai/chapter_multilayer-perceptrons/mlp.html. Acesso em: 06/03/2024.

[10] Uso de Meta-Learning em Tarefas de Aprendizado Profundo. Disponível em: https://www.researchgate.net/figure/Figura-32-Aprendizado-supervisionado-para-o-problema-de-classificacao-de-imagens-O_fig2_363683830. Acesso em: 19/12/2023.

[11] Quais são os tipos de aplicações de Inteligência Artificial mais comuns? Disponível em: https://www.programaria.org/quais-sao-os-tipos-de-aplicacoes-de-inteligencia-artificial-mais-comuns/ . Acesso: 19/11/2023.

[12] Exploring Denoising Autoencoders. Disponível em: https://www.scaler.com/topics/deep-learning/denoising-autoencoder/. Acesso em: 19/11/2023.

[13] Kullback Leibler (KL) Divergence with Examples (Part II): KL Mathematics. Disponível em: https://medium.com/@hosamedwee/kullback-leibler-kl-divergence-with-examples-part-2-9123bff5dc10. Acesso em: 20/11/2023.

[14] Why do we have to normalize the input for an artificial neural network? Disponível em: https://www.geeksforgeeks.org/why-do-we-have-to-normalize-the-input-for-an-artificial-neural-network/. Acesso em: 24/07/2024.

[15] ROOT Reference Guide: Ttree Class Reference. Disponível em: https://root.cern.ch/doc/master/classTTree.html. Acesso em: 26/03/2024.

[16] PyTorch: TORCH.TENSOR. Disponível em: https://pytorch.org/docs/stable/tensors.html. Acesso em 26/03/2024.

[17] ROOT TH1 Class reference, Kolmogorov Test. Disponível em: https://root.cern/doc/master/classTH1.html#aeadcf087afe6ba203bcde124cfabbee4. Acesso em 15/08/2024.