

# Uso do Autoencoder para Detecção de Anomalias NSL-KDD

Joicy Kelly Ferreira de Lara<sup>1</sup>, Heloíse de Souza Bastos<sup>2</sup>, Ruy Oliveira<sup>3</sup>

<sup>1,2,3</sup>Instituto Federal de Mato Grosso - Campus Octayde Jorge da Silva

## Resumo

### RESUMO

**Palavras-chave:** séries temporais; detecção de anomalias; autoencoders; *PyTorch*; dados sintéticos; aprendizado de máquina.

## 1 Introdução

A crescente dependência de sistemas baseados na Internet e redes de computadores torna essencial a implementação de mecanismos robustos de segurança. Com o aumento significativo de dispositivos conectados e serviços baseados na web, o número e a sofisticação dos ataques cibernéticos também têm aumentado, resultando em grandes desafios para a segurança da informação. Entre as principais ameaças estão os ataques de negação de serviço (DoS), invasões de rede e explorações de vulnerabilidades, que podem comprometer a integridade, confidencialidade e disponibilidade dos sistemas (BRONZO, 2023).

Uma abordagem eficaz para combater esses ataques é a detecção de intrusões em redes de computadores, onde técnicas avançadas de aprendizado de máquina têm sido amplamente aplicadas. O uso de sistemas de detecção de intrusão (IDS) permite monitorar e analisar o tráfego de rede em tempo real, identificando padrões anômalos que indicam a presença de possíveis ameaças. Nesse contexto, o dataset NSL-KDD tornou-se uma ferramenta amplamente utilizada para o desenvolvimento e validação de algoritmos de detecção de intrusão, oferecendo dados detalhados sobre tráfego normal e malicioso.

O presente trabalho visa aplicar técnicas de aprendizado de máquina, com foco em autoencoders, para a detecção de anomalias em redes de computadores, utilizando o dataset NSL-KDD. A análise dos dados inclui a identificação de padrões anômalos e normais de tráfego de rede, contribuindo para o aprimoramento de métodos automáticos que permitem detectar ataques de forma mais eficiente e precisa. Este estudo visa explorar a implementação de redes neurais com foco na eficácia de diferentes abordagens para detecção de anomalias, com o objetivo de melhorar a segurança e a resiliência das redes de computadores. A pesquisa foi proposta pelo professor da disciplina de Tópicos de Redes e tem como objetivo proporcionar aos alunos uma compreensão mais profunda sobre o uso de técnicas de aprendizado profundo, como o PyTorch e o autoencoder, aplicadas à detecção de anomalias em redes de computadores.

Este relatório está estruturado da seguinte forma: na Seção 2, discutem-se os conceitos fundamentais que sustentam a implementação do modelo, abrangendo sobre COLOCAR O QUE SERÁ CONTEXTUALIZADO. A Seção 3 apresenta a metodologia usada na detecção de anomalias, enquanto a Seção 4 traz um resultado e discussão sobre a eficácia do modelo. Por fim, na Seção 5, são apresentadas as conclusões.

## 2 Referencial Teórico

A seguir, são abordados os conceitos fundamentais que sustentam a implementação do modelo, como PyTorch, Rede Neural, redes no contexto do dataset, Autoencoder, Autoencoder VAE, detecção de anomalias e tipos de ataques na rede de computadores.

### 2.1 PyTorch

Lançado em 2016 pelo Facebook AI Research (FAIR), o PyTorch é uma biblioteca e framework de código aberto para aprendizado profundo que rapidamente conquistou destaque na comunidade de inteligência artificial (IA)

e aprendizado de máquina (ML). Criado para simplificar a construção de redes neurais, o PyTorch combina a robustez da biblioteca Torch com uma API de alto nível baseada em Python. Sua arquitetura dinâmica e flexível, junto com uma interface intuitiva, possibilita a experimentação e prototipagem ágil, características que o tornaram a escolha preferida entre acadêmicos e pesquisadores. A facilidade de uso, aliada ao suporte contínuo da comunidade, consolidou o PyTorch como uma ferramenta essencial para avanços em áreas como visão computacional, processamento de linguagem natural e redes neurais profundas (SILVA, 2024).

Desenvolvido com base na linguagem de programação Python, o PyTorch é uma biblioteca e framework que oferece uma ampla gama de bibliotecas de modelos pré-configurados e até mesmo modelos já treinados. Essa robustez permite que cientistas de dados construam e implementem redes neurais sofisticadas para deep learning de forma eficiente, reduzindo significativamente o tempo e o esforço necessários em programação e na formulação matemática.

Uma das características mais inovadoras do PyTorch é a sua capacidade de permitir a execução e o teste de partes do código em tempo real. Isso significa que, em vez de aguardar a implementação completa de um modelo, os usuários podem verificar rapidamente se as partes individuais do código estão funcionando corretamente. Essa funcionalidade é especialmente valiosa ao trabalhar com grandes modelos de deep learning, onde a implementação pode ser demorada. Essa abordagem não só promove uma prototipagem rápida, mas também acelera o processo de depuração, permitindo que os cientistas de dados façam ajustes e melhorias de forma mais ágil e eficiente. Como resultado, o PyTorch se destaca como uma plataforma de excelência para a construção e implementação de soluções em inteligência artificial (BERGMANN; STRYKER, 2023).

### 2.1.1 Componentes fundamentais do PyTorch

No PyTorch, tem alguns componentes fundamentais que desempenham papéis essenciais na construção e manipulação de redes neurais (BERGMANN; STRYKER, 2023).

#### 2.1.1.1 Tensores

Os tensores são a estrutura de dados básica do PyTorch. Eles são semelhantes a arrays do NumPy, mas com a adição de suporte a computação em GPUs, o que permite operações mais rápidas e eficientes, especialmente ao lidar com grandes volumes de dados. No contexto do machine learning, um tensor é uma matriz numérica multidimensional que funciona como um dispositivo matemático de contabilidade. De forma linguística, o "tensor" funciona como um termo genérico, incluindo algumas entidades matemáticas mais conhecidas:

- **escalar:** é um tensor zero-dimensional, contendo um único número.
- **vetor:** é um tensor unidimensional, contendo múltiplas escalares do mesmo tipo.
- **tupla:** é um tensor unidimensional contendo diversos tipos de dados.
- **matriz:** é um tensor bidimensional contendo vários vetores do mesmo tipo.
- **tensores N-dimensionais:** tensores com três ou mais dimensões, como os tensores tridimensionais usados para representar imagens RGB em algoritmos de visão computacional

Além de codificar as entradas e saídas de um modelo, os tensores PyTorch também codificam os parâmetros do modelo: os pesos, os vieses e os gradientes que são "aprendidos" no aprendizado de máquina. Esta propriedade de tensores permite a diferenciação automática, uma das características mais importantes do PyTorch.

#### 2.1.1.2 Módulos

No PyTorch, os módulos desempenham um papel fundamental na construção de redes neurais e na implementação de algoritmos de aprendizado profundo. Eles são componentes encapsulados que facilitam a organização e a reutilização do código, tornando o desenvolvimento mais eficiente e estruturado. Existem diferentes tipos de módulos, entre os quais se destacam os módulos `nn`, o módulo `autograd` e os módulos `optim`.

- **Módulos `nn`:** Os módulos `nn` (ou `torch.nn`) são essenciais para a criação de redes neurais em PyTorch. Essa biblioteca fornece classes que representam diferentes camadas de redes neurais, como camadas lineares, convolucionais, de pooling, de normalização, entre outras. Cada um desses módulos herda da classe base `nn.Module`, que permite que os desenvolvedores definam sua própria arquitetura de rede de maneira intuitiva e eficiente.

Por exemplo, ao construir uma Rede Neural Convolucional (CNN), um desenvolvedor pode combinar várias camadas `nn.Conv2d`, `nn.ReLU` e `nn.MaxPool2d` para formar um pipeline de processamento de imagem. Além disso, os módulos `nn` facilitam a implementação do método `forward`, que define como os dados

são processados através das camadas. Essa estrutura modular não apenas simplifica o design, mas também torna o código mais legível e fácil de manter.

- **Módulo autograd:** O módulo `autograd` é uma das características mais poderosas do PyTorch, permitindo a diferenciação automática. Isso significa que o PyTorch pode calcular gradientes automaticamente durante o treinamento de modelos de aprendizado profundo. Quando uma operação é realizada em tensores que têm o atributo `requires_grad` definido como `True`, o PyTorch registra essa operação em um grafo computacional.

Durante a retropropagação, o `autograd` utiliza esse grafo para calcular os gradientes de forma eficiente, permitindo ajustes nos pesos da rede neural de acordo com a função de perda. Essa capacidade de diferenciação automática não apenas simplifica a implementação de algoritmos de aprendizado, mas também torna o PyTorch uma escolha preferida entre pesquisadores e profissionais de IA, já que elimina a necessidade de derivar manualmente as funções de perda.

- **Módulos Optim:** Os módulos `optim` ou `torch.optim` são responsáveis pela implementação de algoritmos de otimização utilizados para atualizar os parâmetros da rede neural durante o treinamento. Este módulo oferece uma variedade de otimizadores, como `SGD` (*Stochastic Gradient Descent*), `Adam` (*Adaptive Moment Estimation*), `RMSprop` (*Root Mean Square Propagation*), entre outros, cada um com suas particularidades e vantagens.

Os otimizadores trabalham em conjunto com o `autograd`, utilizando os gradientes calculados para ajustar os pesos da rede de forma que a função de perda seja minimizada. A escolha do otimizador pode ter um impacto significativo no desempenho do modelo, e os pesquisadores frequentemente experimentam diferentes algoritmos para encontrar a melhor combinação de velocidade de convergência e precisão.

#### 2.1.1.3 Gráficos Dinâmicos de Computação

Os Gráficos Dinâmicos de Computação (DCGs) são uma característica essencial do PyTorch, permitindo a representação flexível de modelos de deep learning. Diferentemente dos gráficos de computação estática, que requerem a definição completa da rede antes da execução, os DCGs possibilitam modificações em tempo real, facilitando a depuração e a prototipagem. Essa flexibilidade é particularmente útil durante o treinamento, pois permite ajustes dinâmicos durante a retropropagação. Enquanto gráficos estáticos podem oferecer eficiência em certos casos, eles são limitados em sua capacidade de adaptação a diferentes formatos de dados, como em redes neurais convolucionais (CNNs) que processam imagens de tamanhos variados (BERGMANN; STRYKER, 2023).

Em resumo, os DCGs no PyTorch promovem uma abordagem prática e inovadora na construção de redes neurais, otimizando o desenvolvimento e o treinamento de modelos complexos, sendo uma escolha popular entre acadêmicos e desenvolvedores na área de inteligência artificial.

#### 2.1.1.4 Diferenciação automática

A diferenciação automática é um método essencial para treinar redes neurais, especialmente em aprendizado supervisionado, utilizando o algoritmo de `backpropagation`. Durante a passagem adiante, um modelo é alimentado com entradas e gera previsões, das quais se calcula um erro com uma função de perda. A partir daí, a diferenciação é usada para obter a derivada dessa função, permitindo o ajuste dos pesos da rede neural por meio do gradiente descendente, camada por camada.

O módulo `autograd` do PyTorch implementa a diferenciação automática utilizando a regra da cadeia, o que simplifica o cálculo de derivados complexos ao dividi-los em componentes mais simples. Ele registra automaticamente os gradientes para todas as operações em um gráfico computacional, reduzindo significativamente a necessidade de cálculos manuais na `backpropagation`. Quando um modelo treinado é executado, o uso do `autograd` se torna desnecessário. Para otimizar o uso de recursos computacionais, pode-se desativar o rastreamento de gradientes definindo `requires_grad=False` em operações de tensor, indicando ao PyTorch que não deve mais registrar gradientes (DIFERENCIAÇÃO..., 2022).

#### 2.1.1.5 Conjuntos de dados e carregadores de dados

Trabalhar com grandes conjuntos de dados é um dos principais desafios no treinamento de modelos de deep learning, pois exige gerenciamento eficiente de dados e poder computacional significativo. O PyTorch oferece duas estruturas fundamentais que simplificam essa tarefa: `Dataset` e `Dataloader`, ambas localizadas em `torch.utils.data`.

- **Dataset:** A classe `torch.utils.data.Dataset` armazena as amostras e seus rótulos associados, servindo como uma interface base para conjuntos de dados. Para usá-la, você define como os dados serão lidos e organizados. Normalmente, cria-se uma subclasse `Dataset` personalizada que implementa os métodos `__len__` (para obter o número total de amostras) e `__getitem__` (para acessar uma amostra específica com base no índice). Essa estrutura é altamente flexível e permite organizar tanto conjuntos de dados simples quanto complexos, como imagens, texto e dados multimodais.
- **Dataloader:** Envolvendo o objeto `Dataset`, o `torch.utils.data.DataLoader` cria um iterável sobre o conjunto de dados que facilita o carregamento e o processamento de amostras, especialmente em lotes (*batches*), para melhorar a eficiência computacional. Com a capacidade de realizar operações como empacotamento, divisão em lotes e multiprocessamento, o `Dataloader` simplifica o fluxo de dados durante o treinamento. Ele permite que cada lote de dados seja carregado de forma assíncrona, reduzindo o tempo ocioso entre as etapas de treinamento e aumentando a velocidade do modelo.

Essas duas primitivas trabalham juntas para garantir que o pipeline de dados seja contínuo, eficiente e fácil de ajustar, adaptando-se bem a diversas necessidades de armazenamento e formatos de dados usados em projetos de deep learning.

## 2.2 Redes Neurais

As Redes Neurais Artificiais (RNAs) representam modelos computacionais inspirados na estrutura e no funcionamento do sistema nervoso biológico, particularmente no cérebro humano, com o objetivo de imitar o aprendizado e a tomada de decisão de seres inteligentes. Estruturalmente, as RNAs são compostas por diversas unidades de processamento, conhecidas como neurônios artificiais, que simulam as células neurais do cérebro, chamadas de neurônios biológicos. Cada neurônio biológico é composto por dendritos, que recebem informações; um corpo celular, que processa esses dados; e um axônio, que transmite a resposta para outros neurônios. Esse modelo de comunicação é o que inspirou a criação das redes neurais no campo da computação (PREGOWSKA; OSIAL, 2022).

No contexto das RNAs, os neurônios artificiais interagem de maneira semelhante, recebendo e processando informações em camadas e aprendendo com a experiência, ou seja, através de um processo de ajuste de "pesos" nas conexões, que representam a força da comunicação entre as unidades. Redes neurais artificiais podem variar em tamanho e complexidade, com algumas redes contendo centenas ou milhares de neurônios artificiais, enquanto o cérebro humano contém bilhões. Esse ajuste contínuo, inspirado pelo aprendizado do cérebro, permite que redes neurais sejam capazes de resolver problemas complexos em áreas como visão computacional, processamento de linguagem natural e reconhecimento de padrões.

As redes neurais artificiais (ANNs) são um método de inteligência artificial inspirado no funcionamento do cérebro humano. Utilizando uma estrutura em camadas composta por "nós" ou neurônios artificiais interconectados, as redes neurais fazem parte de uma subcategoria do aprendizado de máquina conhecida como aprendizado profundo (deep learning). O objetivo dessas redes é criar sistemas adaptativos que permitam aos computadores aprender com erros e aprimorar-se continuamente. Esses sistemas são compostos por uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída, representando uma arquitetura que permite processar dados de forma complexa e autônoma (AWS, 2023).

### 2.2.1 Arquitetura de uma Rede Neural

O cérebro humano é a inspiração por trás da arquitetura da rede neural. As células do cérebro humano, chamadas de neurônios, formam uma rede altamente interconectada e enviam sinais elétricos entre si para ajudar os seres humanos a processar informações. Da mesma forma, uma rede neural artificial é feita de neurônios artificiais que trabalham juntos para resolver um problema. Os neurônios artificiais são módulos de software chamados de "nós", e as redes neurais artificiais são programas ou algoritmos de software que, em seu núcleo, usam sistemas de computação para solucionar cálculos matemáticos (AWS, 2023).

Cada nó na rede possui um peso e um limiar, fatores que determinam se o nó será ativado ou não. Quando a saída do nó ultrapassa o valor de limiar, ele é ativado, transmitindo informações para a camada seguinte; caso contrário, não repassa dados adiante. Esse processo é essencial para o ajuste de precisão, permitindo que a rede aprenda a identificar e classificar dados com alta exatidão. Durante o treinamento, as redes neurais ajustam esses pesos e limiares, otimizando-se a partir dos dados e aprimorando a precisão dos resultados ao longo do tempo (IBM, 2023b).

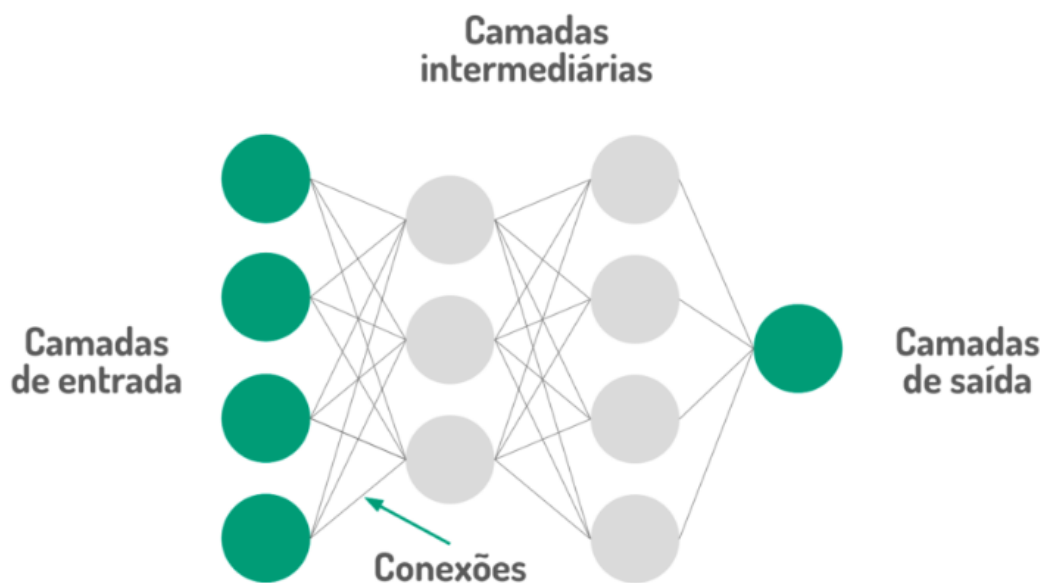
A arquitetura de uma rede neural é a estrutura organizacional de seus neurônios artificiais em diferentes camadas para processar dados e realizar tarefas de aprendizado. Em uma rede neural simples, três camadas principais interagem para transformar dados de entrada em previsões ou classificações.

- **Camada de entrada:** A camada de entrada é o ponto inicial para os dados que entram na rede neural. Cada nó ou neurônio na camada de entrada representa uma característica específica dos dados (por exemplo, pixels de uma imagem, palavras de um texto, ou valores de sensores). Essa camada não realiza processamento; ela apenas transmite as informações para as próximas camadas.
- **Camada Oculta:** Após a camada de entrada, os dados passam para uma ou mais camadas ocultas. As camadas ocultas são onde ocorrem as operações matemáticas principais. Cada neurônio em uma camada oculta aplica uma função de ativação a uma combinação ponderada dos valores recebidos da camada anterior. Isso permite que a rede detecte padrões complexos e relacione dados de entrada a características específicas. Quanto mais camadas ocultas uma rede possui, mais complexa será sua arquitetura, permitindo que ela reconheça padrões mais profundos, como em redes neurais profundas (ou deep learning), que possuem múltiplas camadas ocultas.
- **Camada de Saída:** A camada de saída gera o resultado final do processo da rede neural. No caso de uma classificação binária, a camada de saída pode ter um único nó, retornando um valor binário (por exemplo, 0 ou 1). Em tarefas de classificação com múltiplas categorias, a camada de saída terá um nó para cada classe possível, representando a probabilidade de cada classe com base nas entradas.

Redes neurais profundas (ou deep neural networks) contêm múltiplas camadas ocultas, o que as torna capazes de aprender padrões complexos e realizar tarefas sofisticadas, como reconhecimento de imagem e processamento de linguagem natural. Cada conexão entre neurônios possui um peso, determinando a força da influência entre um nó e outro. Esses pesos são ajustados durante o treinamento para minimizar o erro nas previsões da rede. No entanto, redes profundas demandam grandes volumes de dados para evitar o problema de overfitting (quando o modelo se ajusta demais aos dados de treinamento, comprometendo sua precisão em novos dados) e podem ser exigentes em termos de recursos computacionais (AWS, 2023).

Essa organização em camadas, junto com o ajuste dos pesos e o uso de funções de ativação, permite que redes neurais sejam ferramentas poderosas na modelagem de problemas complexos e no avanço de várias aplicações de inteligência artificial.

Figura 1: Estrutura de uma Rede Neural



Fonte: (BIOLOGIX, 2024)

### 2.2.2 Tipos de Redes Neurais

As redes neurais artificiais (ANNs) possuem diversas arquiteturas, cada uma voltada para resolver tipos específicos de problemas. Os principais tipos incluem:

- **Perceptron Multicamadas (MLP - *Multilayer Perceptron*):** Consiste em camadas de entrada, ocultas e saída, onde cada nó é totalmente conectado aos nós da camada seguinte. É amplamente utilizada para tarefas de classificação e regressão.
- **Redes Neurais Convolucionais (CNN - *Convolutional Neural Networks*):** Composta por camadas de convolução, pooling e completamente conectadas, as CNNs são usadas principalmente em visão computacional, reconhecimento de imagem e processamento de vídeo. Elas detectam padrões espaciais ao aplicar filtros às imagens.
- **Redes Neurais Recorrentes (RNN - *Recurrent Neural Networks*):** Projetadas para processar sequências de dados, as RNNs têm conexões recorrentes, o que lhes permite reter informações de etapas anteriores. São eficazes em tarefas como tradução automática e análise de séries temporais. Uma variante popular é a *Long Short-Term Memory (LSTM)*, que lida com problemas de "desvanecimento de gradiente" comuns nas RNNs tradicionais.
- **Redes Adversariais Generativas (GAN - *Generative Adversarial Networks*):** Composta por uma rede geradora e uma discriminadora, as GANs são utilizadas para criar dados novos e realistas. São amplamente aplicadas em geração de imagens, arte digital e síntese de voz.
- **Redes Neurais de Memória Curta e Longa (LSTM):** Uma subcategoria das RNNs, as LSTMs foram projetadas para resolver o problema de longa dependência em dados sequenciais. Elas são especialmente úteis em processamento de linguagem natural e análise de séries temporais de longo prazo.
- **Redes Neurais de Base Radial (RBF - *Radial Basis Function Networks*):** Utilizam funções de base radial como funções de ativação, sendo particularmente úteis em sistemas de classificação. Elas calculam a distância de entrada em relação a um ponto de referência para determinar a saída.
- **Transformers:** Baseadas em camadas de atenção, os Transformers revolucionaram o processamento de linguagem natural ao permitir o processamento paralelo de dados sequenciais. Modelos como BERT e GPT são exemplos populares, sendo aplicados em tarefas como tradução, resumo e geração de texto.
- **Autoencoders:** Redes neurais projetadas para compressão de dados, capturando as características mais importantes dos dados de entrada. São utilizadas em redução de dimensionalidade, detecção de anomalias e aprendizado não supervisionado.

Esses tipos de redes neurais se diferenciam nas arquiteturas e aplicabilidades, e a escolha depende das especificidades da tarefa e do tipo de dado a ser processado. Cada tipo de rede neural traz avanços em campos distintos, como reconhecimento de imagem, processamento de linguagem natural e geração de conteúdo. Neste trabalho será usado a Rede Neural Autoencoder.

### 2.2.3 Hiperparâmetros e componentes de configuração

Os conceitos a seguir são, em grande parte, hiperparâmetros e componentes de configuração das redes neurais e do processo de treinamento de modelos de aprendizado de máquina. Cada um desses elementos influencia diretamente a capacidade do modelo de aprender padrões nos dados e a qualidade do seu desempenho em novas amostras (GUARDIEIRO, 2021).

- **Funções de Ativação**  
Definem como a soma ponderada dos neurônios é transformada antes de passar para a próxima camada. Algumas funções comuns incluem:
  - **ReLU (Rectified Linear Unit)** É rápida e eficiente para a maioria das redes profundas, ajudando a reduzir problemas de gradiente.
  - **Sigmoid e Tanh:** Utilizadas historicamente, mas sofrem com problemas de saturação de gradiente.
  - **Softmax:** Geralmente usada na camada de saída em tarefas de classificação multiclases.
  - **Leaky ReLU e ELU:** Variações de ReLU que tratam problemas de dying neurons.
- **Quantidade de Amostras e Tamanho do Conjunto de Dados**
  - **Quantidade de Amostras (Samples):** O número de amostras no conjunto de dados de treino e teste pode impactar diretamente a qualidade do modelo. Mais dados geralmente resultam em melhor desempenho.
  - **Tamanho dos Lotes (Batch Size):** Define quantas amostras a rede processa antes de atualizar os pesos. Tamanhos menores podem melhorar a regularização, enquanto tamanhos maiores podem acelerar o treinamento.

- **Função de Perda (Loss Function)**

Calcula o erro entre a previsão da rede e a resposta esperada. A escolha depende do tipo de problema:

- **Cross-Entropy Loss:** Usada para problemas de classificação.
- **Mean Squared Error (MSE):** Comum em problemas de regressão.
- **Binary Cross-Entropy:** Usada para classificações binárias.

- **Otimizadores**

Método para ajustar os pesos da rede a fim de minimizar a função de perda. Otimizadores comuns incluem:

- **SGD (Stochastic Gradient Descent):** Otimizador básico, que pode ser melhorado com momentum.
- **Adam (Adaptive Moment Estimation):** Combina momentos de gradiente e é eficiente para a maioria das redes.
- **RMSprop:** Uma variante de SGD, bem-sucedida em redes recorrentes.

- **Taxa de Aprendizado (Learning Rate)**

Controla o tamanho dos passos de ajuste dos pesos. Taxas muito altas ou baixas podem dificultar o aprendizado.

- **Regularização**

- **Dropout:** Zera aleatoriamente uma fração dos neurônios durante o treinamento para evitar overfitting.
- **L1/L2 Regularização:** Penaliza pesos grandes para evitar que o modelo se ajuste demais aos dados de treino.
- **Batch Normalization:** Normaliza as ativações dentro de cada lote, acelerando o treinamento e estabilizando o aprendizado.

- **Parâmetros de Treinamento e Avaliação**

- **Número de Épocas (Epochs):** Quantas vezes o modelo verá todo o conjunto de dados durante o treinamento.
- **Métricas de Avaliação:** Avaliam o desempenho do modelo durante e após o treinamento, como acurácia, precisão, recall, F1-score, etc.

- **Épocas**

Em aprendizado de máquina, uma época refere-se a uma única passagem por todo o conjunto de dados de treinamento durante o processo de treinamento de um modelo. Em outras palavras, uma época é quando o algoritmo de aprendizado atualiza os pesos do modelo uma vez com base em todos os exemplos de treinamento(ZADROZNY, 2021). Aqui estão alguns pontos importantes sobre épocas:

- **Ciclo Completo:** Durante uma época, o modelo vê cada exemplo de treinamento uma vez.
- **Várias Épocas:** O treinamento geralmente envolve várias épocas (por exemplo, 50, 100, 150, etc.) para permitir que o modelo aprenda padrões nos dados. Cada época pode melhorar a performance do modelo à medida que ele ajusta seus parâmetros.
- **Overfitting:** Se o número de épocas for muito alto, o modelo pode começar a "decorar" os dados de treinamento, resultando em overfitting, onde o desempenho no conjunto de teste diminui.
- **Validação:** Muitas vezes, o desempenho do modelo é avaliado em um conjunto de validação após cada época para monitorar o progresso e decidir quando parar o treinamento.

#### 2.2.4 Métodos de aprendizado de máquina

O aprendizado supervisionado, não supervisionado e o semissupervisionado são três abordagens principais no campo de aprendizado de máquina, com objetivos e metodologias distintas para treinar modelos de inteligência artificial (IA) em diferentes tipos de tarefas.

- **Aprendizado Supervisionado:** O aprendizado supervisionado envolve o treinamento de modelos com dados rotulados, ou seja, cada entrada no conjunto de dados possui uma saída esperada ou rótulo conhecido. Isso permite que o modelo aprenda a relação entre as entradas e as saídas desejadas, facilitando o processo de prever rótulos corretos para novos dados. Durante o treinamento supervisionado, o modelo ajusta seus parâmetros para minimizar o erro entre suas previsões e os rótulos reais, um processo que

geralmente utiliza algoritmos como a regressão linear, as redes neurais e as árvores de decisão (O... , 2022).

O aprendizado supervisionado é especialmente útil para problemas em que o objetivo é bem definido e possui resultados previsíveis, como a classificação de imagens (ex.: identificar se uma imagem contém um gato ou um cachorro) ou a previsão de valores numéricos (ex.: prever o preço de uma casa com base em suas características). No entanto, uma desvantagem do aprendizado supervisionado é a necessidade de grandes volumes de dados rotulados, que podem ser difíceis e caros de coletar e anotar.

- **Aprendizado Não Supervisionado:** O aprendizado não supervisionado, por outro lado, é utilizado quando o modelo trabalha com dados sem rótulos, o que significa que ele precisa identificar padrões ou relacionamentos ocultos sem orientação explícita. Nesse caso, o modelo busca estruturas, como grupos ou categorias naturais, dentro dos dados por meio de métodos como a clusterização e a redução de dimensionalidade. Os algoritmos comuns para aprendizado não supervisionado incluem o *k-means* (para agrupamento) e a análise de componentes principais (PCA, para redução de dimensionalidade) (O... , 2022).

Esta abordagem é ideal para tarefas em que não há rótulos definidos ou quando o objetivo é explorar os dados, identificar grupos ou encontrar associações. Um exemplo típico de aprendizado não supervisionado é a segmentação de clientes em marketing, onde a empresa busca entender padrões de comportamento do consumidor para adaptar campanhas. No entanto, a avaliação do aprendizado não supervisionado pode ser mais complexa, pois o modelo não possui uma “resposta correta” a ser comparada.

Ambos os métodos possuem vantagens e desvantagens. O aprendizado supervisionado é eficiente em tarefas com respostas claras e pode alcançar alta precisão quando os dados são rotulados adequadamente. Já o aprendizado não supervisionado oferece flexibilidade e é capaz de lidar com grandes volumes de dados não rotulados, o que o torna aplicável a diversas áreas de pesquisa e análise exploratória de dados.

## 2.3 Dataset NSL - KDD

O conjunto de dados NSL-KDD é uma base de dados popular para a detecção de intrusões em redes, amplamente utilizada para pesquisa e desenvolvimento de modelos de segurança cibernética e *machine learning*. A sigla NSL-KDD significa *Network Security Laboratory - Knowledge Discovery and Data Mining*. Esse conjunto de dados foi criado pelo *Network Security Laboratory* (NSL) da Universidade de New Brunswick, no Canadá, como uma versão aprimorada do conjunto *KDD Cup 1999* para pesquisas em detecção de intrusões. A base KDD original, que servia para detectar ataques e intrusões, apresentava redundâncias e desequilíbrios significativos nos dados, o que prejudicava a precisão dos modelos desenvolvidos para detecção de intrusões (M, 2018).

O NSL-KDD aborda essas limitações ao reduzir as redundâncias e balancear o número de amostras. Ele contém dados de tráfego de rede e classifica cada conexão em tipos de ataque ou como tráfego normal. Os dados de NSL-KDD contêm 41 atributos, que representam aspectos da conexão de rede, como duração, número de pacotes enviados, tipo de protocolo, serviço solicitado, *bytes* de origem e destino, entre outros. O conjunto classifica os registros em 40 diferentes classes, a classe normal, que representa as conexões normais e inofensivas e outras 39 classes, que representam as conexões maliciosas. As conexões maliciosas podem ser divididas em 4 categorias de ataque:

- DoS (*Denial of Service*)
- Probe
- R2L (*Remote to Local*)
- U2R (*User to Root*)

Por fornecer uma base de dados rotulada e com uma divisão equilibrada entre ataques e tráfego legítimo, o NSL-KDD é amplamente utilizado no treinamento e avaliação de algoritmos de detecção de intrusões em redes, especialmente em cenários onde a análise em tempo real é importante. Isso inclui sistemas de *Intrusion Detection System* (IDS), onde o NSL-KDD ajuda a simular e detectar atividades anômalas em redes com mais eficiência.

Embora o NSL-KDD represente uma evolução em relação ao *KDD Cup 1999*, ainda possui limitações, como a falta de atualizações para tipos de ataques mais recentes. No entanto, ele continua sendo uma base fundamental na área de cibersegurança e aprendizado de máquina, fornecendo uma plataforma consistente para a criação e comparação de métodos de detecção de anomalias. O NSL-KDD pode ser encontrado para download e análise



no Kaggle, onde é amplamente utilizado pela comunidade acadêmica e de pesquisa para aprimorar métodos de segurança digital em redes.

Por este trabalho tratar de detecção de anomalias, a classificação dos registros será dividida em apenas duas classes, normal e anormal.

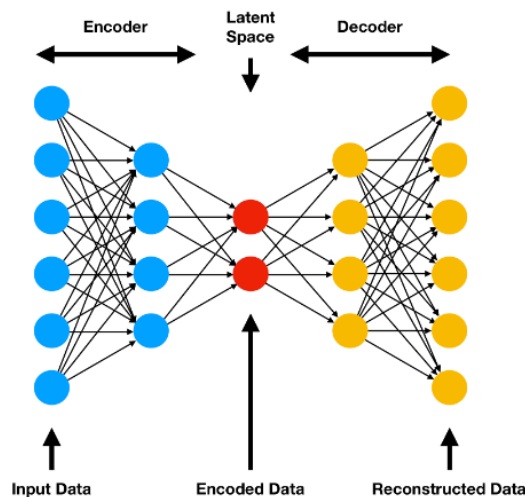
## 2.4 Autoencoders

Os *autoencoders* são uma classe especial de redes neurais artificiais amplamente utilizadas para tarefas de compressão de dados e aprendizado não supervisionado, que é o modelo treinado com dados não rotulados, ou seja, não há rótulos associados aos dados. O modelo precisa encontrar padrões ou estruturas nos dados por conta própria, já a supervisionada é modelo treinado com dados rotulados, ou seja, cada entrada tem uma saída esperada associada (rótulo). A principal característica de um autoencoder é sua estrutura composta por duas partes: o codificador (*encoder*) e o decodificador (*decoder*), que trabalham juntos para transformar os dados de entrada em uma representação compacta e, em seguida, tentar reconstruí-los a partir dessa representação (Geeks for Geeks, 2023).

### 2.4.1 Estrutura de um Autoencoder

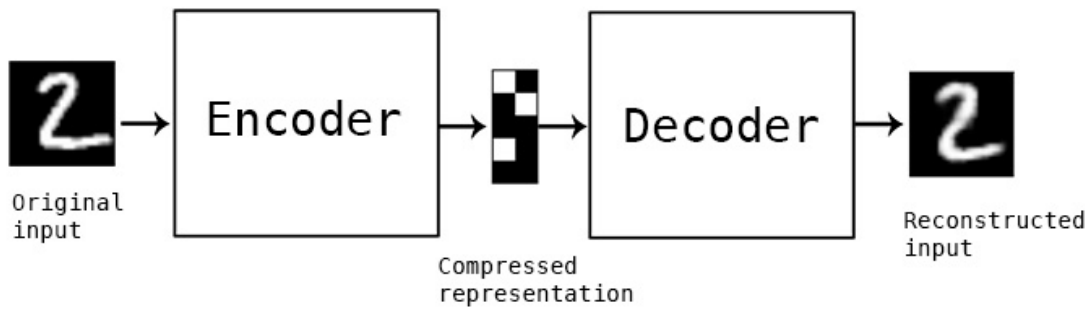
- **Encoder:** O codificador é responsável por comprimir os dados de entrada em uma representação de menor dimensionalidade. Essa compactação é feita em camadas sucessivas, reduzindo a dimensionalidade do conjunto de dados. O objetivo dessa parte da rede é capturar as características mais relevantes da entrada, removendo ruídos e informações redundantes.;
- **Latent Space:** O ponto de compressão máxima entre o codificador e o decodificador é conhecido como "espaço latente". Esse espaço é uma representação de baixa dimensionalidade que retém as informações essenciais da entrada original.
- **Decoder:** O decoder realiza a tarefa inversa, tentando reconstruir a entrada original a partir da representação latente gerada pelo encoder. O sucesso desse processo depende da capacidade do modelo de aprender os padrões relevantes nos dados.

Figura 2: Diagrama do Autoencoder



Fonte: (STEVEN, 2019).

Figura 3: Demonstração do funcionamento do autoencoder



Fonte: (CHOLLET, 2016)

#### 2.4.2 Função de Custo

O treinamento do *autoencoder* é orientado pela função de custo, que geralmente é o erro quadrático médio (*MSE* - *Mean Squared Error*). O MSE mede a diferença entre a entrada original e a saída reconstruída pelo decodificador. O objetivo do treinamento é minimizar essa diferença, garantindo que a rede seja capaz de comprimir e descomprimir os dados sem perder informações importantes.

#### 2.4.3 Aplicações dos Autoencoders

Os *autoencoders* têm uma ampla gama de aplicações, incluindo:

- **Redução de Dimensionalidade:** Utilizados para reduzir a complexidade dos dados, preservando características essenciais, como uma forma mais eficiente de compressão em relação às abordagens tradicionais de redução de dimensionalidade, como a Análise de Componentes Principais (PCA);
- **Detecção de Anomalias:** Uma das principais aplicações, como neste trabalho, é a detecção de anomalias. Como o *autoencoder* é treinado para reconstruir dados que seguem um padrão, ele terá dificuldades em reconstruir dados anômalos ou fora desse padrão, resultando em um erro de reconstrução elevado;
- **Filtragem de Ruídos:** Outra aplicação importante dos *autoencoders* é a remoção de ruídos de dados, como em imagens ou sinais, para melhorar a qualidade da informação.

### 2.5 Autoencoder VAE - Variational Autoencoder

O Autoencoder Variacional (VAE), ou *Variational Autoencoder*, é um tipo de rede neural projetado para aprendizado não supervisionado e geração de dados. É especialmente útil para compressão de informações e criação de amostras realistas a partir de dados complexos, como imagens e sinais. Diferente de um autoencoder tradicional, que aprende uma representação fixa dos dados, o VAE combina redes neurais e modelos probabilísticos para aprender uma representação latente contínua e estruturada dos dados, o que facilita a criação de novas amostras que sejam semelhantes, mas não idênticas, aos dados originais (IBM, 2023a).

A arquitetura de um VAE é construída de modo a aliar aprendizado profundo e estatísticas probabilísticas para modelar dados complexos, sendo assim uma extensão avançada do autoencoder tradicional. No VAE, o espaço latente aprendido é probabilístico, representando não apenas uma única posição para cada dado, mas uma distribuição latente contínua que permite o uso de amostras da distribuição para gerar variações do dado de entrada original (RAMOS, 2023).

Essa representação probabilística no espaço latente permite uma série de operações que não são possíveis em autoencoders convencionais. Por exemplo, ao modelar uma distribuição de probabilidade, o VAE pode amostrar diretamente do espaço latente para criar dados realistas, gerar transições suaves entre diferentes amostras e interpolar características. Assim, o VAE consegue tanto comprimir eficientemente quanto gerar novas amostras, com características que imitam as dos dados de entrada originais, mas com um toque de variabilidade e flexibilidade que permite a criação de novos dados semelhantes ao original (BOOK, 2022).

A capacidade do VAE de capturar uma estrutura contínua dos dados no espaço latente o torna popular em várias aplicações, incluindo a geração de imagens realistas, compressão de dados, criação de amostras sintéticas,

interpolação de amostras e até mesmo a detecção de anomalias em conjuntos de dados. O uso do VAE revolucionou o campo de aprendizado profundo e modelagem probabilística, contribuindo significativamente para a geração de conteúdo realista e manipulação de dados complexos em áreas como visão computacional e processamento de sinais.

O modelo é composto por uma etapa de codificação, amostragem e decodificação, e o treinamento envolve os seguintes passos:

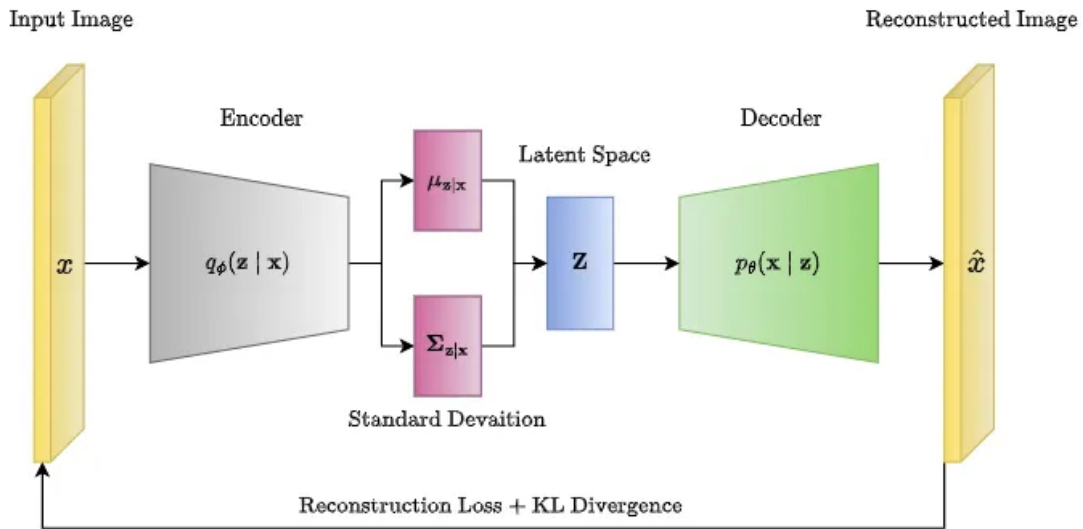
- **Codificação:** O modelo passa os dados de entrada por duas camadas ocultas, aplicando ativações ReLU para aprender uma representação reduzida. A saída da última camada de codificação fornece uma média ( $\mu$ ) e uma log variância ( $\log\sigma^2$ ), que descrevem uma distribuição latente.
- **Amostragem (Reparametrização):** Com a média e a variância, o modelo gera uma "amostra" a partir da distribuição latente. A reparametrização (adicionando uma pequena variação) permite que a rede seja treinável por retropropagação.
- **Decodificação:** O vetor latente amostrado passa por camadas para ser reconstruído, usando uma ativação sigmoid na saída para obter valores na mesma escala dos dados originais.

- **Função de perda**

A perda do VAE é uma combinação de:

- **Erro de reconstrução (MSE):** mede a diferença entre o dado original e o reconstruído. Divergência KL (KLD), que regula o espaço latente para se aproximar de uma distribuição normal padrão, garantindo que o modelo possa gerar amostras variadas.
- **Otimização e Treinamento:** Em cada época, o modelo processa os dados em lotes, calcula a perda, e ajusta os pesos através da retropropagação, usando o otimizador Adam. A pequena taxa de aprendizado e o uso de dropout ajudam a evitar overfitting.

Figura 4: Autocodificador Variacional (VAE)

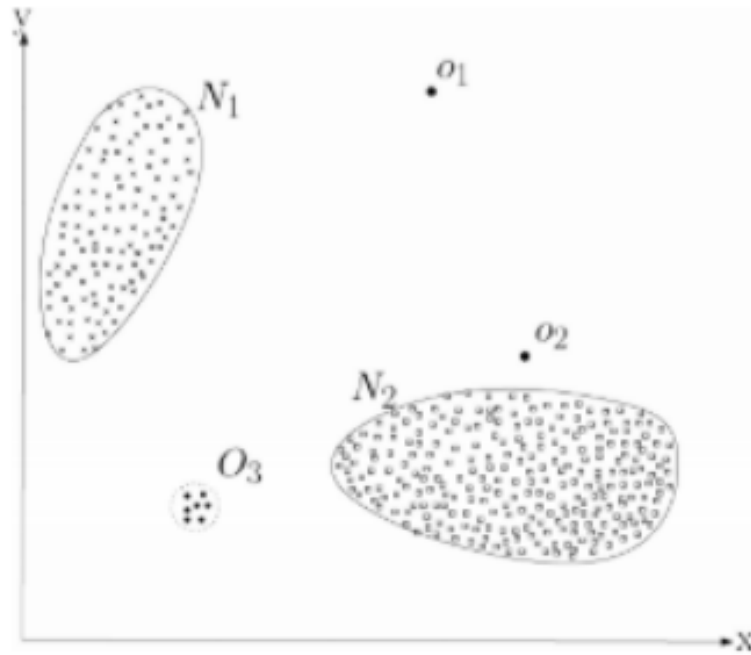


Fonte: (SHENDE, 2023).

## 2.6 Detecção de anomalias

A detecção de anomalias consiste em identificar eventos ou observações que desviam significativamente dos padrões observados em dados normais. A abordagem tradicional para detecção de anomalias envolve o uso de técnicas estatísticas que, embora eficazes para dados simples, falham ao lidar com conjuntos de dados de alta dimensionalidade e complexidade estrutural, como imagens, sinais de sensores e séries temporais. Redes neurais, especialmente autoencoders, surgem como uma solução para esses casos, uma vez que conseguem aprender representações latentes úteis de dados complexos, comprimindo informações em um espaço menor e destacando características que ajudam na diferenciação de comportamentos normais e anômalos (COMINI, 2019).

Figura 5: Exemplo simples de anomalias em um espaço bidimensional.



Fonte: (COMINI, 2019).

Na Figura 5, podemos observar um exemplo de anomalias em um espaço bidimensional, onde pontos com valores incomuns ou anômalos aparecem claramente distantes dos grandes aglomerados de dados, que representam os valores normais. Esses pontos isolados, ao se afastarem significativamente dos padrões centrais, indicam uma diferença estrutural em relação à maioria dos dados, permitindo a identificação visual de anomalias com base na distância em relação aos clusters principais.

### 2.6.1 Detecção de Anomalias usando VAE

A metodologia para detecção de anomalias com VAE segue três etapas principais:

- **Treinamento com Dados Normais:**

Primeiramente, o VAE é treinado com dados que representam o comportamento esperado do sistema, como leituras de sensores ou registros de transações normais. Durante o treinamento, a rede aprende a modelar a distribuição dos dados normais, capturando as variações permitidas dentro desse comportamento. Esse treinamento permite que o modelo aprenda os padrões e as características que definem o comportamento esperado.

- **Reconstrução de Novas Amostras**

Após o treinamento, o VAE é utilizado para reconstruir novas amostras de dados, tanto normais quanto anômalas. Como o modelo foi treinado exclusivamente com dados normais, ele consegue reconstruir essas amostras com um erro pequeno. No entanto, quando uma amostra que não se ajusta aos padrões normais (uma anomalia) é submetida ao VAE, o modelo apresenta um erro de reconstrução maior, uma vez que essa amostra não se alinha com a distribuição latente aprendida.

- **Avaliação do Erro de Reconstrução e Definição de Limiar**

Para a detecção de anomalias, é necessário definir um limiar de erro de reconstrução. Esse limiar representa o valor acima do qual uma amostra é considerada anômala. Pode ser estabelecido empiricamente ou por métodos estatísticos, de forma a maximizar a precisão e minimizar os falsos positivos. Assim, quando o erro de reconstrução de uma nova amostra excede o limiar, essa amostra é marcada como uma anomalia.

### 2.6.2 Vantagens do Uso do VAE para Detecção de Anomalias

O uso de VAEs para detecção de anomalias oferece várias vantagens sobre métodos tradicionais e autoencoders determinísticos:

- **Modelagem de Variações Naturais:** o VAE aprende uma distribuição latente contínua, permitindo que o modelo represente variações normais nos dados sem identificá-las incorretamente como anomalias.

- **Capacidade de Generalização:** como o VAE gera uma distribuição latente para cada dimensão, ele generaliza bem em dados complexos e de alta dimensionalidade, adaptando-se a contextos variados.
- **Detecção Não Supervisionada:** o VAE é ideal para contextos onde há poucos dados anômalos rotulados, pois pode ser treinado exclusivamente com dados normais.
- **Eficiência Computacional:** apesar da complexidade da modelagem probabilística, o VAE é computacionalmente eficiente em comparação com modelos adversariais e permite a implementação em tempo real.

## 2.7 Tipos de ataques na redes de computadores

Ataques em redes de computadores são ações maliciosas projetadas para comprometer a integridade, confidencialidade e disponibilidade dos sistemas, dados e serviços conectados à rede. Estes ataques podem variar em natureza, complexidade e propósito, abrangendo desde a interceptação de dados até a interrupção de operações. Entender os diferentes tipos de ataques é crucial para desenvolver estratégias de defesa e proteção eficazes (SILVA, 2018). Abaixo, são descritos os principais tipos de ataques que comumente afetam redes de computadores:

- **Ataques de Negação de Serviço (DoS e DDoS)**  
Os ataques de Negação de Serviço (*DoS - Denial of Service*) e Negação de Serviço Distribuída (DDoS - Distributed Denial of Service) têm como objetivo sobrecarregar um recurso ou serviço, tornando-o indisponível para seus usuários legítimos. Em um ataque DoS, um único sistema é utilizado para inundar o alvo com requisições excessivas. No caso do DDoS, múltiplos sistemas comprometidos, muitas vezes controlados remotamente por meio de redes de bots, são coordenados para amplificar o ataque, tornando-o mais difícil de detectar e combater. Esses ataques são comuns contra websites, serviços online e infraestruturas críticas.
- **Ataque Probe**  
Os ataques do tipo Probe (ou escaneamento) visam mapear a estrutura e o estado da rede para identificar sistemas vulneráveis. Esses ataques envolvem uma série de varreduras e sondagens que coletam informações sobre a rede, como portas abertas, sistemas operacionais em uso e serviços ativos. Por meio dessa análise, o atacante pode descobrir potenciais alvos para outros tipos de ataque. Ataques Probe são considerados preparatórios e, embora não prejudiquem diretamente os recursos da rede, podem permitir ataques mais destrutivos se um invasor identificar pontos fracos. Ferramentas de escaneamento, como o Nmap, são amplamente usadas para conduzir esses ataques.
- **Ataque R2L (*Remote-to-Local*)**  
O ataque *Remote-to-Local* (R2L) ocorre quando um atacante externo tenta obter acesso não autorizado a uma máquina da rede como um usuário local. Neste tipo de ataque, o invasor geralmente não possui privilégios no sistema e usa técnicas de invasão para explorar vulnerabilidades ou fraquezas na autenticação para ganhar acesso. Os ataques R2L podem incluir técnicas de força bruta para adivinhar senhas, phishing ou o uso de explorações de software que permitam acesso remoto não autorizado. Uma vez dentro do sistema, o atacante pode obter dados confidenciais ou até tentar escalar privilégios para acessar áreas mais restritas.
- **Ataque U2R (*User-to-Root*)**  
O ataque *User-to-Root* (U2R) é uma tentativa de um usuário comum ou com privilégios limitados de obter acesso administrativo ou de "root" em um sistema. Esse tipo de ataque aproveita vulnerabilidades de escalonamento de privilégio, que permitem que um usuário autorizado ou invasor eleve seus privilégios e obtenha o controle total do sistema. Ataques U2R geralmente envolvem exploração de falhas no software, como buffer overflows, falhas de configuração ou problemas de permissão. Um invasor que consiga executar um ataque U2R com sucesso poderá modificar arquivos do sistema, instalar malware ou manter o controle do sistema comprometido.

Cada um desses ataques representa uma ameaça única à segurança de redes, explorando diferentes níveis de acesso e objetivos específicos. Enquanto o DoS afeta a disponibilidade dos serviços, o Probe foca em descobrir vulnerabilidades. Já os ataques R2L e U2R envolvem tentativas de invasão e escalonamento de privilégios, respectivamente, para obter acesso e controle dos sistemas. Para enfrentar esses ataques, é fundamental que sistemas e redes adotem estratégias de defesa como firewalls, sistemas de detecção de intrusão (IDS), autenticação forte e práticas de monitoramento contínuo, além de atualizações regulares de software e treinamentos de conscientização de segurança (SILVA, 2018).

### 3 Metodologia

A metodologia adotada neste trabalho para a detecção de anomalias em redes de computadores foi baseada na implementação de um Variational Autoencoder (VAE) utilizando a biblioteca *PyTorch*. O processo foi estruturado em várias etapas, conforme detalhado a seguir:

#### 3.1 Seleção do Conjunto de Dados

O conjunto de dados utilizado foi o NSL-KDD, um benchmark amplamente reconhecido para a detecção de intrusões e anomalias em redes. A escolha desse dataset foi realizada pelo professor, que destacou sua relevância na área de segurança cibernética, além de sua variedade de ataques e comportamentos normais. O dataset foi dividido em duas partes: 70% dos dados foram utilizados para o treinamento do modelo, enquanto os 30% restantes foram reservados para a validação do desempenho do modelo.

#### 3.2 Instalação de Bibliotecas Necessárias

Antes de iniciar a implementação, foi necessário instalar diversas bibliotecas que são essenciais para o processamento e a manipulação dos dados, além do treinamento da rede neural. As seguintes bibliotecas foram instaladas:

Figura 6: Bibliotecas instaladas.

```
%pip install opendatasets
%pip install seaborn
%pip install seaborn matplotlib
%pip install scipy
%pip install keras
%pip install torch
```

Fonte: (Autoria própria, 2024).

Essas bibliotecas incluem:

- **Open Datasets:** Para facilitar o download de conjuntos de dados da internet.
- **Seaborn e Matplotlib:** Para visualização de dados.
- **SciPy:** Para funcionalidades científicas adicionais.
- **Keras:** Embora não tenha sido o foco principal do projeto, foi instalado para casos de comparação futura.
- **Torch:** Biblioteca fundamental para a construção e treinamento de modelos de aprendizado profundo.

Após a instalação deve ser feito a importação das bibliotecas necessárias e o download do conjunto de dados.

Figura 7: Download do conjunto de dados

```
# Faz o download do dataset # URL do dataset no Kaggle
pd.download("https://www.kaggle.com/datasets/hassan06/nsllkdd/data")
```

Fonte: (Autoria própria, 2024).

#### 3.3 Análise Exploratória de Dados

Antes de treinar o modelo, foi realizada uma análise exploratória de dados (AED) para entender melhor as características do conjunto de dados NSL-KDD. Essa análise incluiu:

- **Visualização da Distribuição dos Dados:** Utilizando gráficos de dispersão, histogramas e gráficos de densidade para identificar a distribuição dos dados e detectar possíveis outliers. As bibliotecas Seaborn e Matplotlib foram fundamentais para essa etapa.

- Identificação de Classes: A AED permitiu uma melhor compreensão das classes presentes no conjunto de dados, incluindo a proporção de amostras normais e de anomalias. Essa informação é crucial para garantir que o modelo não aprenda a priorizar uma classe em detrimento da outra.
- Normalização dos Dados: Foi aplicada a normalização dos dados utilizando o MinMaxScaler para garantir que todas as features estivessem na mesma escala, o que é especialmente importante para o treinamento de redes neurais.
- Tratamento de Dados Faltantes: Verificou-se a existência de valores ausentes.

Figura 8: Informações sobre o tamanho do conjunto de dados

```
# Carregando os conjuntos de dados de treinamento e teste
train_df = pd.read_csv('nslkdd/KDDTrain+.txt', header=None, names=columns)
test_df = pd.read_csv('nslkdd/KDDTest+.txt', header=None, names=columns)

# Concatenando os DataFrames para criar um único DataFrame
df = pd.concat([train_df, test_df], ignore_index=True)

# Exibindo os tamanhos dos conjuntos resultantes
print(f"Tamanho do Conjunto de Dados Completo: {len(df)}")
print(f"Tamanho do Conjunto de Treinamento: {len(train_df)}")
print(f"Tamanho do Conjunto de Teste: {len(test_df)}")
```

Tamanho do Conjunto de Dados Completo: 148517  
Tamanho do Conjunto de Treinamento: 125973  
Tamanho do Conjunto de Teste: 22544

Fonte: (Autoria própria, 2024).

Figura 9: Exibição dos tipos de classes

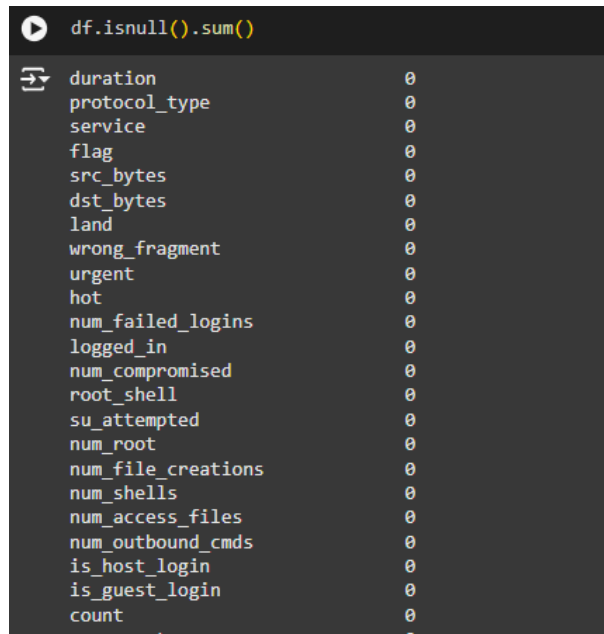
```
train_df['class_type'].value_counts()
```

class_type	count
normal	67343
neptune	41214
satan	3633
ipsweep	3599
portsweep	2931
smurf	2646
nmap	1493
back	956
teardrop	892
warezclient	890
pod	201
guess_passwd	53
buffer_overflow	30
warezmaster	20
land	18
imap	11
rootkit	10
loadmodule	9
ftp_write	8
multihop	7
phf	4
perl	3
spy	2

Name: count, dtype: int64

Fonte: (Autoria própria, 2024).

Figura 10: Verificação dos dados nulos



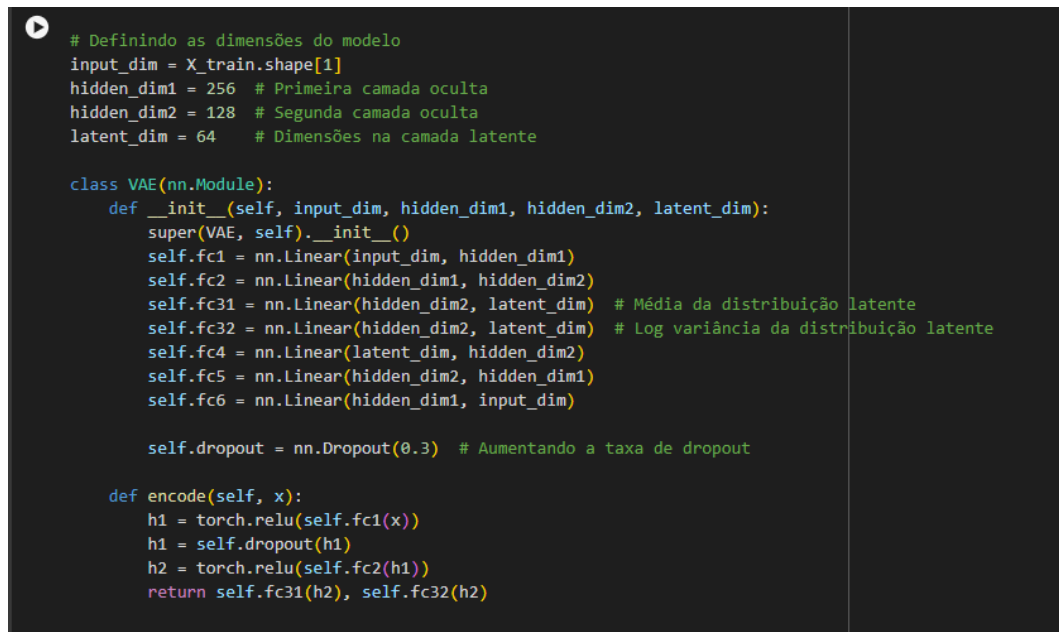
duration	0
protocol_type	0
service	0
flag	0
src_bytes	0
dst_bytes	0
land	0
wrong_fragment	0
urgent	0
hot	0
num_failed_logins	0
logged_in	0
num_compromised	0
root_shell	0
su_attempted	0
num_root	0
num_file_creations	0
num_shells	0
num_access_files	0
num_outbound_cmds	0
is_host_login	0
is_guest_login	0
count	0

Fonte: (Autoria própria, 2024).

### 3.4 Implementação da Arquitetura do VAE

A arquitetura do VAE foi implementada em PyTorch, consistindo de uma camada de entrada que representa a dimensão dos dados do NSL-KDD, uma camada oculta com 256 unidades, uma segunda camada oculta com 128 unidades e uma camada latente com 64 unidades. Essa estrutura permite ao modelo aprender representações significativas dos dados, capturando a complexidade necessária para a detecção de anomalias.

Figura 11: Implementação do Autoencoder VAE



```
# Definindo as dimensões do modelo
input_dim = X_train.shape[1]
hidden_dim1 = 256 # Primeira camada oculta
hidden_dim2 = 128 # Segunda camada oculta
latent_dim = 64 # Dimensões na camada latente

class VAE(nn.Module):
    def __init__(self, input_dim, hidden_dim1, hidden_dim2, latent_dim):
        super(VAE, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim1)
        self.fc2 = nn.Linear(hidden_dim1, hidden_dim2)
        self.fc31 = nn.Linear(hidden_dim2, latent_dim) # Média da distribuição latente
        self.fc32 = nn.Linear(hidden_dim2, latent_dim) # Log variância da distribuição latente
        self.fc4 = nn.Linear(latent_dim, hidden_dim2)
        self.fc5 = nn.Linear(hidden_dim2, hidden_dim1)
        self.fc6 = nn.Linear(hidden_dim1, input_dim)

        self.dropout = nn.Dropout(0.3) # Aumentando a taxa de dropout

    def encode(self, x):
        h1 = torch.relu(self.fc1(x))
        h1 = self.dropout(h1)
        h2 = torch.relu(self.fc2(h1))
        return self.fc31(h2), self.fc32(h2)

    def decode(self, z):
        h2 = torch.relu(self.fc4(z))
        h1 = torch.relu(self.fc5(h2))
        x_reconstructed = self.fc6(h1)
        return x_reconstructed
```

Fonte: (Autoria própria, 2024).



### 3.5 Configuração do Treinamento

A configuração do treinamento incluiu a definição de uma função de perda composta pela soma da Erro Quadrático Médio (MSE) e da Divergência Kullback-Leibler (KLD). A função de perda foi implementada conforme mostrado abaixo:

Figura 12: Função de perda usando MSE e KLD

```
# Função de perda usando BCE e KLD
def loss_function(recon_x, x, mu, logvar):
    BCE = nn.functional.mse_loss(recon_x, x, reduction='sum') # Usando MSE
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return BCE + KLD
```

Fonte: (Autoria própria, 2024).

O treinamento foi realizado por 20 épocas, com um otimizador Adam ajustado para uma taxa de aprendizado de 5e-5 e um decaimento de peso de 1e-5. O loop de treinamento é ilustrado a seguir:

Figura 13: Número de épocas e otimizador

```
# Aumentando o número de épocas e ajustando o otimizador
num_epochs = 20
optimizer = torch.optim.Adam(model.parameters(), lr=5e-5, weight_decay=1e-5)

# Treinamento
for epoch in range(num_epochs):
    model.train()
    train_loss = 0
    for batch_idx, (data,) in enumerate(train_loader):
        optimizer.zero_grad()
        data = data.view(-1, input_dim)
        recon_batch, mu, logvar = model(data)
        loss = loss_function(recon_batch, data, mu, logvar)
        loss.backward()
        train_loss += loss.item()
        optimizer.step()

    train_loss /= len(train_loader.dataset)
    print(f'Epoch {epoch + 1} Loss: {train_loss:.4f}')
```

Fonte: (Autoria própria, 2024).

### 3.6 Avaliação do Modelo

A validação do modelo é uma etapa crucial no processo de desenvolvimento de redes neurais, pois permite avaliar o desempenho e a eficácia do modelo na detecção de anomalias. Para este estudo, utilizamos um modelo de Autoencoder Variacional (VAE) treinado com o conjunto de dados NSL-KDD, focando na identificação de atividades anômalas em redes. A validação foi realizada em um conjunto de dados separado, que não foi utilizado durante o treinamento, garantindo que as métricas de desempenho refletissem a capacidade real do modelo de generalizar.

A primeira parte da validação envolveu o cálculo do erro de reconstrução, que foi obtido através da comparação entre os dados de entrada originais e os dados reconstruídos pelo modelo. A função de erro utilizada foi o erro quadrático médio (MSE), que fornece uma medida quantitativa da diferença entre os valores reais e os valores previstos.

Em seguida, definimos um limiar (threshold) para classificar as instâncias como anômalas ou normais. Se o erro de reconstrução excedesse esse limiar, a instância seria classificada como anômala. Essa abordagem de limiar permite uma flexibilidade na detecção, que pode ser ajustada de acordo com os requisitos do sistema.

Por fim, a acurácia do modelo foi calculada, comparando as previsões do modelo com os rótulos verdadeiros das instâncias de teste. A acurácia fornece uma métrica simples, mas eficaz, para avaliar o desempenho do modelo, indicando a proporção de previsões corretas em relação ao total de previsões feitas.

## 4 Resultados e Discussões

O modelo de rede neural desenvolvido para a detecção de anomalias apresentou uma acurácia de 73%, um resultado que indica que, em média, o modelo é capaz de classificar corretamente 73% das instâncias testadas. Essa taxa de acerto, embora relativamente positiva, suscita uma análise mais aprofundada sobre o desempenho do modelo, especialmente considerando o contexto específico da aplicação.

Ao examinar as predições do modelo, notou-se que uma parte significativa das classificações realizadas consistiu em prever a classe majoritária, que, neste caso, é a classe "normal". Isso levanta a possibilidade de um desbalanceamento nas classes, onde o modelo pode estar se aproveitando da predominância de exemplos normais para alcançar uma acurácia aparentemente aceitável. Para uma avaliação mais completa, seria prudente considerar a construção de uma matriz de confusão, que permitiria observar quantos verdadeiros positivos, verdadeiros negativos, falsos positivos e falsos negativos o modelo produziu. Essa análise é crucial para entender o real desempenho do modelo em identificar anomalias, uma vez que a alta acurácia pode, em alguns casos, mascarar um desempenho insatisfatório na detecção de eventos críticos.

Além disso, a escolha do threshold (limite) para classificar as anomalias tem um impacto significativo nas métricas de desempenho. Um threshold inadequado pode resultar em um aumento no número de falsos negativos, comprometendo a eficácia do modelo em identificar verdadeiras anomalias. A análise de outros indicadores, como precisão, recall e F1-score, pode proporcionar uma visão mais holística do desempenho do modelo.

Os resultados obtidos também abrem espaço para possíveis melhorias. A otimização de hiperparâmetros, o aprimoramento da qualidade e da quantidade de dados de treinamento, bem como a exploração de diferentes arquiteturas de modelos, são caminhos viáveis que podem levar a um desempenho superior. A implementação de técnicas de aumento de dados e a experimentação com outras abordagens de machine learning podem contribuir para a criação de um modelo mais robusto e eficaz na detecção de anomalias.

## 5 Considerações finais

Em suma, o modelo de rede neural desenvolvido demonstrou um desempenho inicial promissor com uma acurácia de 73%, mas a interpretação desse resultado exige cautela. A análise detalhada das predições e a consideração do desbalanceamento nas classes são fundamentais para uma avaliação mais precisa da eficácia do modelo. Para que o modelo atinja um desempenho mais robusto, recomenda-se a implementação de ajustes e melhorias nas estratégias de treinamento, incluindo a análise do threshold, a utilização de métricas complementares e a exploração de diferentes arquiteturas. Ao adotar essas abordagens, o modelo pode se tornar mais confiável e eficaz na detecção de anomalias, atendendo melhor às necessidades práticas de sua aplicação específica.

## Referências

AWS. **O que é uma rede neural?** AWS, 2023. [S. l.]. Acesso em: 28 out. 2024. Disponível em: <https://aws.amazon.com/pt/what-is/neural-network/>.

BERGMANN, D.; STRYKER, C. **O que é PyTorch?** 2023. Acesso em: 28 out. 2024. Disponível em: <https://www.ibm.com/br-pt/topics/pytorch>.

BIOLOGIX. **Redes Neurais: uma vertente da Inteligência Artificial inspirada no funcionamento do cérebro humano.** 2024. [S. l.]. Acesso em: 28 out. 2024. Disponível em: <https://www.biologix.com.br/2024/07/19/redes-neurais-uma-vertente-da-inteligencia-artificial-inspirada-no-funcionamento-do-cerebro-humano/>.

BOOK, D. L. **CAPÍTULO 60 – Variational Autoencoders (VAEs) – Definição, Redução de Dimensionalidade, Espaço Latente e Regularização.** 2022. Acesso em: 28 out. 2024. Disponível em: <https://www.deeplearningbook.com.br/variational-autoencoders-vaes-definicao-reducao-de-dimensionalidade-espaco-latente-e-regularizacao/>.

BRONZO, W. Trabalho de Conclusão (Graduação Engenharia de Computação), **Utilizando máquinas de vetor de suporte para detecção de ataques de rede**. [S. l.]: [s.n.], 2023. Acesso em: 18 out. 2024. Disponível em: <https://repositorio.ufrn.br/bitstream/123456789/55909/1/UtilizandoMaquinasdeVetordeSuporte-Bronzo.2023.pdf>.

CHOLLET, F. **Building Autoencoders in Keras**. 2016. [S. l.]. Disponível em: <https://blog.keras.io/building-autoencoders-in-keras.html>.

COMINI, J. P. M. **Detecção de anomalias utilizando autoencoder variacional**. 2019. Trabalho de Conclusão (Graduação em Ciência da Computação) - Universidade Estadual Paulista "Júlio de Mesquita Filho". Disponível em: <https://repositorio.unesp.br/server/api/core/bitstreams/7e48d35b-3699-4286-8849-cfc768e57a88/content>.

DIFERENCIAÇÃO Automática. 2022. [S. l.: s. n.]. Acesso em: 28 out. 2024. Disponível em: [https://pt.d2l.ai/chapter\\_preliminaries/autograd.html#:~:text=A%20diferencia%C3%A7%C3%A3o%20autom%C3%A1tica%20permite%20que,em%20rela%C3%A7%C3%A3o%20a%20cada%20par%C3%A2metro](https://pt.d2l.ai/chapter_preliminaries/autograd.html#:~:text=A%20diferencia%C3%A7%C3%A3o%20autom%C3%A1tica%20permite%20que,em%20rela%C3%A7%C3%A3o%20a%20cada%20par%C3%A2metro).

Geeks for Geeks. **AUTOENCODERS - Machine Learning**. 2023. Acesso em: 3 out. 2024. Disponível em: <https://www.geeksforgeeks.org/auto-encoders/>.

GUARDIEIRO, A. **Tudo que você já deveria saber sobre otimização de hiperparâmetros em redes neurais — Parte I**. 2021. Acesso em: 28 out. 2024. Disponível em: <https://medium.com/datarisk-io/tudo-que-voc%C3%AA-j%C3%A1-deveria-saber-sobre-otimiza%C3%A7%C3%A3o-de-hiperpar%C3%A2metros-em-redes-neurais-parte-i-f1d8975f0177>.

IBM. **O que é um codificador automático?** 2023. Acesso em: 28 out. 2024. Disponível em: <https://www.ibm.com/br-pt/topics/autoencoder>.

IBM. **O que é uma rede neural?** IBM, 2023. [S. l.]. Acesso em: 28 out. 2024. Disponível em: <https://www.ibm.com/br-pt/topics/neural-networks>.

M, H. Z. **NSL-KDD**. 2018. [S. l.]. Acesso em: 28 out. 2024. Disponível em: <https://www.kaggle.com/datasets/hassan06/nslkdd>.

O que é aprendizado de máquina (ML)? 2022. [S. l.]. Acesso em: 28 out. 2024. Disponível em: <https://www.ibm.com/br-pt/topics/machine-learning>.

PREGOWSKA, A.; OSIAL, M. **O que é uma rede neural e para que serve?** 2022. [S. l.]. Acesso em: 28 out. 2024. Disponível em: <https://parajovens.unesp.br/o-que-e-uma-rede-social-e-para-que-serve/>.

RAMOS, M. **O que é Variational Autoencoder (Autoencoder Variacional – VAE)?** 2023. Acesso em: 28 out. 2024. Disponível em: <https://glossario.maiconramos.com/glossario/o-que-e-variational-autoencoder-autoencoder-variacional-vae/>.

SHENDE, R. **Autocodificadores, Autocodificadores Variacionais (VAE) e -VAE**. 2023. Acesso em: 28 out. 2024. Disponível em: <https://medium.com/@rushikesh.shende/autoencoders-variational-autoencoders-vae-and-%CE%B2-vae-ceba9998773d>.

SILVA, A. J. **Afinal, o que é Pytorch?** 2024. Acesso em: 28 out. 2024. Disponível em: <https://medium.com/@alexjosesilva/afinal-o-que-%C3%A9-pytorch-ale90c00ed12>.

SILVA, H. D. d. **Detecção de ataques de negação de serviço em redes de computadores através de previsões por séries temporais**. 2018. Dissertação (Mestre em Modelagem Computacional em Ciência e Tecnologia) - Universidade Federal Fluminense. Disponível em: <http://mcct.uff.br/wp-content/uploads/sites/454/2019/10/Disserta%C3%A7%C3%A3o-Henrique-Dornel-da-Silva-defendeu-em-15-08-2018.pdf>.

STEVEN, F. **Variational Autoencoders are Beautiful**. 2019. Accessed: October 4, 2024. Disponível em: <https://www.compthree.com/blog/autoencoder/>.

ZADROZNY, B. **Métodos Estatísticos de Aprendizagem**. 2021. Acesso em: 28 out. 2024. Disponível em: <http://profs.ic.uff.br/~bianca/ia/aulas/IA-Aula21.pdf>.