

Inteligência Artificial

Raoni F. S. Teixeira

Aula 13 - Aprendizado Não-supervisionado: Clustering k -means e Autoencoder

1 Introdução

O objetivo do aprendizado não supervisionado é identificar padrões em conjuntos de dados que não possuem rótulos ou respostas pré-definidas. Diferentemente do aprendizado supervisionado, não há uma saída específica para guiar o treinamento.

Nesta aula, vamos estudar dois métodos: k -means e autoencoder, que abordam problemas como agrupamento e redução de dimensionalidade.

2 Algoritmo k -means

K -means é um algoritmo de *clustering* que recebe um conjunto de dados de treinamento $\{x^{(1)}, \dots, x^{(m)}\}$ e busca agrupá-los em k subconjuntos coesos chamados de *clusters*. Cada subconjunto j é representado pelo seu centróide μ_j . O algoritmo pode ser descrito pelos seguintes passos:

1. Inicialize os centróides $\mu_1, \mu_2, \dots, \mu_k \in \mathbb{R}^n$ aleatoriamente.
2. Repita até a convergência:
 - Para cada exemplo i , atualize a atribuição do cluster:

$$c^{(i)} := \arg \min_j \|x^{(i)} - \mu_j\|^2.$$

- Para cada centróide j , atualize sua posição:

$$\mu_j := \frac{\sum_{i=1}^m \mathbb{1}\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^m \mathbb{1}\{c^{(i)} = j\}}.$$

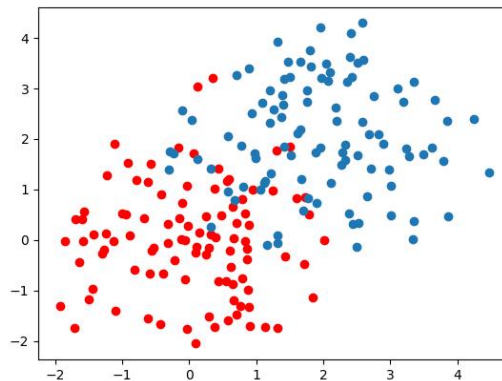


Figura 1: Distribuição gaussiana com duas classes. Vermelho: $\mu_1 = [0, 0]$. Azul: $\mu_2 = [2, 2]$.

No algoritmo acima, k é um parâmetro que representa o número de clusters desejado, enquanto os centróides μ_j indicam as posições estimadas dos centros desses clusters. A inicialização dos centróides pode ser feita escolhendo aleatoriamente k exemplos de treinamento e definindo os centróides iguais aos valores desses exemplos.

A Figura 1 mostra dois grupos de pontos gerados por distribuições gaussianas: pontos vermelhos com média $\mu_1 = [0, 0]$ e azuis com média $\mu_2 = [2, 2]$, ambos com matriz de covariância $C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

A Figura 2 mostra o algoritmo k -means aplicado aos dados da Figura 1. A cada iteração, o algoritmo alterna entre duas etapas:

1. Atribuição: cada ponto é associado ao centróide mais próximo
2. Atualização: os centróides são recalculados como a média dos pontos associados

Após convergência, os centróides ($c_1 = [-0.003, -0.009]$ e $c_2 = [2.071, 2.143]$) aproximam-se das médias originais.

O algoritmo minimiza a função de custo:

$$J = \sum_{i=1}^k \sum_{x \in C_i} \|x - \mu_i\|^2,$$

em que C_i corresponde aos pontos do cluster i , μ_i é o centróide do cluster i e $\|x - \mu_i\|^2$ é a distância euclidiana ao quadrado.

A Figura 3 mostra a função de custo e a convergência do algoritmo em cada iteração. A medida que os centróides se aproximam dos valores iniciais (Figura 2 a)-f)) a função de custo diminui (Figura 3).

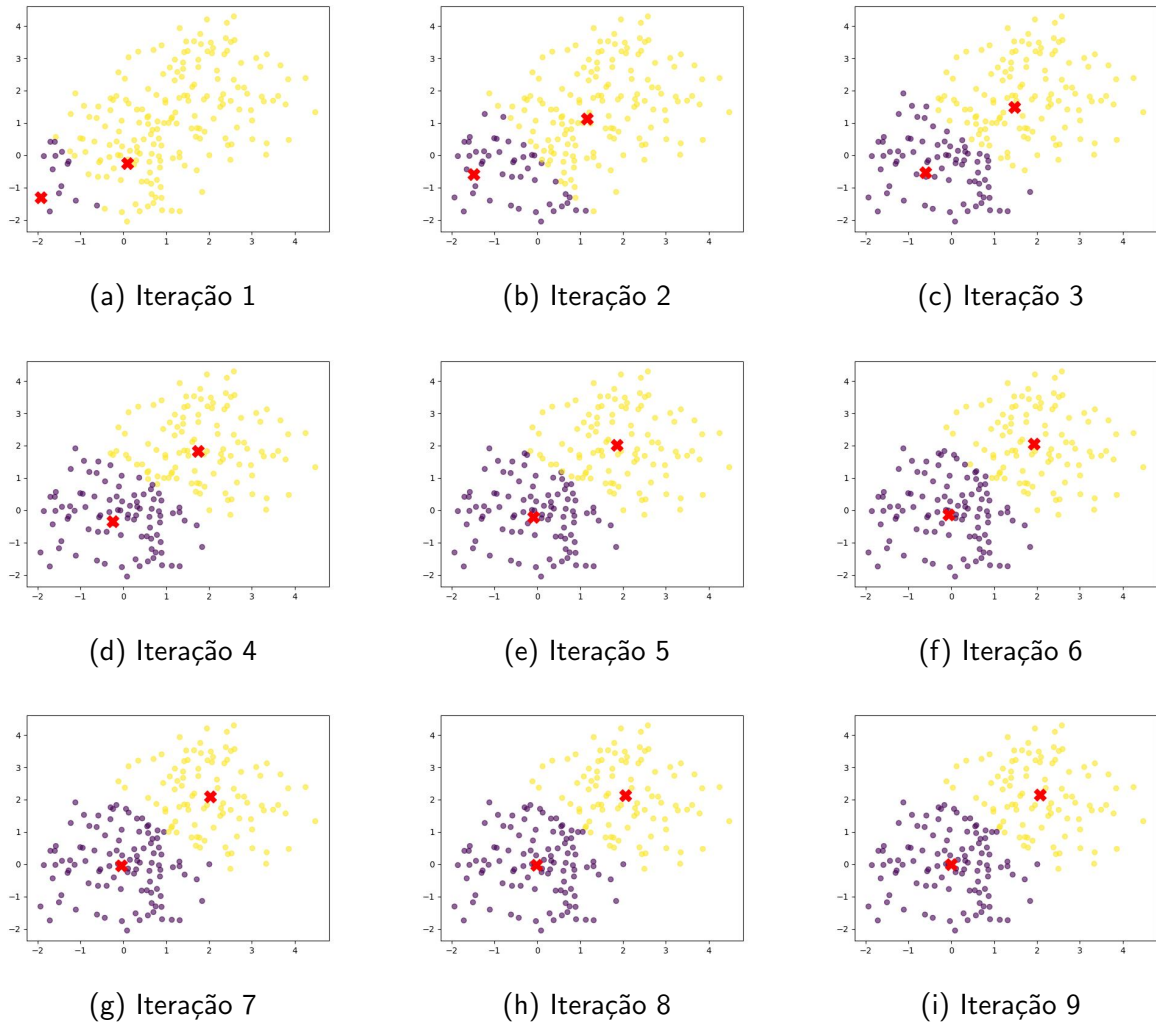


Figura 2: Evolução do k -means em nove iterações. Cruzes vermelhas indicam centróides. Os valores finais aproximam-se das médias originais.

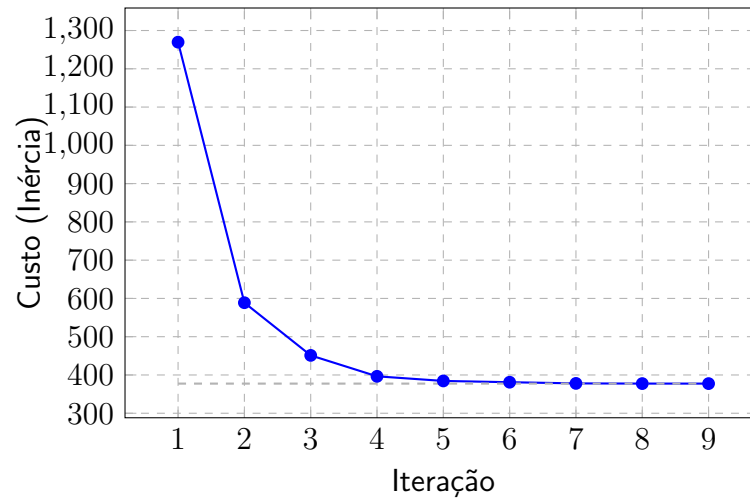


Figura 3: Convergência do K-Means: Função de Custo ao longo das iterações.

2.1 Aplicações

No mundo real, o k -means é amplamente utilizado em aplicações de segmentação de mercado. Por exemplo, uma fábrica de camisas pode utilizar o algoritmo para agrupar clientes com base em medidas corporais como circunferência do tronco e comprimento dos braços.

A Figura 4 mostra um agrupamento de pessoas em tamanhos de camisas (P, M, G) com base nessas medidas. As cruzes correspondem aos centróides. Ao escolher cuidadosamente os parâmetros do algoritmo, a fábrica pode garantir que a maior parte da população será atendida com seus tamanhos padrão.

O k -means também pode ser utilizado para reduzir a resolução de imagens através de quantização de cores. A Figura 5 mostra um exemplo desse processo usando diferentes valores de k (32, 16, 8, 4 e 2 cores).

O algoritmo funciona da seguinte forma:

- Agrupa as cores da imagem original em k clusters
- Substitui cada pixel pelo centróide do seu cluster correspondente
- Gera uma nova imagem com apenas k cores distintas

Esse processo reduz efetivamente a complexidade da imagem enquanto mantém suas características principais.

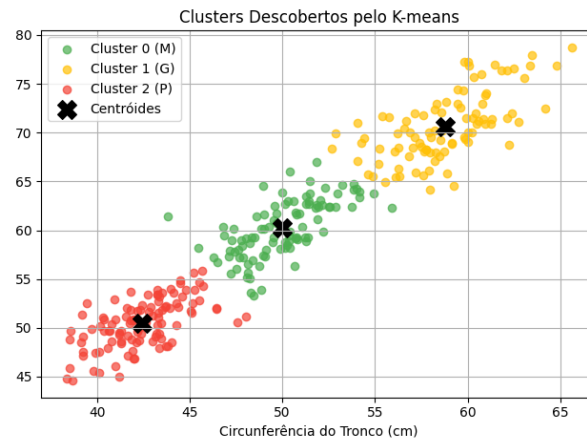


Figura 4: Agrupamento de tamanhos de camisas usando k -means com base em medidas corporais

3 Autoencoders

Um autoencoder é uma rede neural que aprende representações compactas de dados de forma não supervisionada. Seu funcionamento baseia-se em dois processos:

- **Codificação:** Transforma os dados em uma representação compacta (espaço latente) e
- **Decodificação:** Reconstrói os dados originais a partir dessa versão comprimida.

A Figura 6 mostra como esses dois processos são incorporados na arquitetura do modelo. O **Encoder** é uma rede neural que reduz progressivamente a dimensionalidade da entrada (por exemplo, $784 \rightarrow 256 \rightarrow 32$). O **Espaço Latente** é a representação da entrada original compactada $\mathbf{z} \in \mathbb{R}^K$. K é muito menor que D . Por fim, o **Decoder** é a rede que reconstitui os dados a partir do espaço latente \mathbf{z} – a dimensão aumenta progressivamente, por exemplo, $32 \rightarrow 256 \rightarrow 784$.

O autoencoder aprende minimizando o erro de reconstrução:

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^n \|\mathbf{x}_i - \text{Decoder}(\text{Encoder}(\mathbf{x}_i))\|^2, \quad (1)$$

em que \mathbf{x}_i representa os dados originais e n o número de amostras. Essa função de custo mede a perda de informação durante o processo de compressão e reconstrução. Quanto menor o valor de \mathcal{L} mais precisa é a reconstrução $\text{Decoder}(\text{Encoder}(\mathbf{x}_i))$.

Tanto o encoder quanto o decoder (representados pelos trapézios vermelhos na Figura 6) podem ser implementados com diversos componentes neurais — desde camadas densas até arquiteturas convolucionais ou *transformers*.

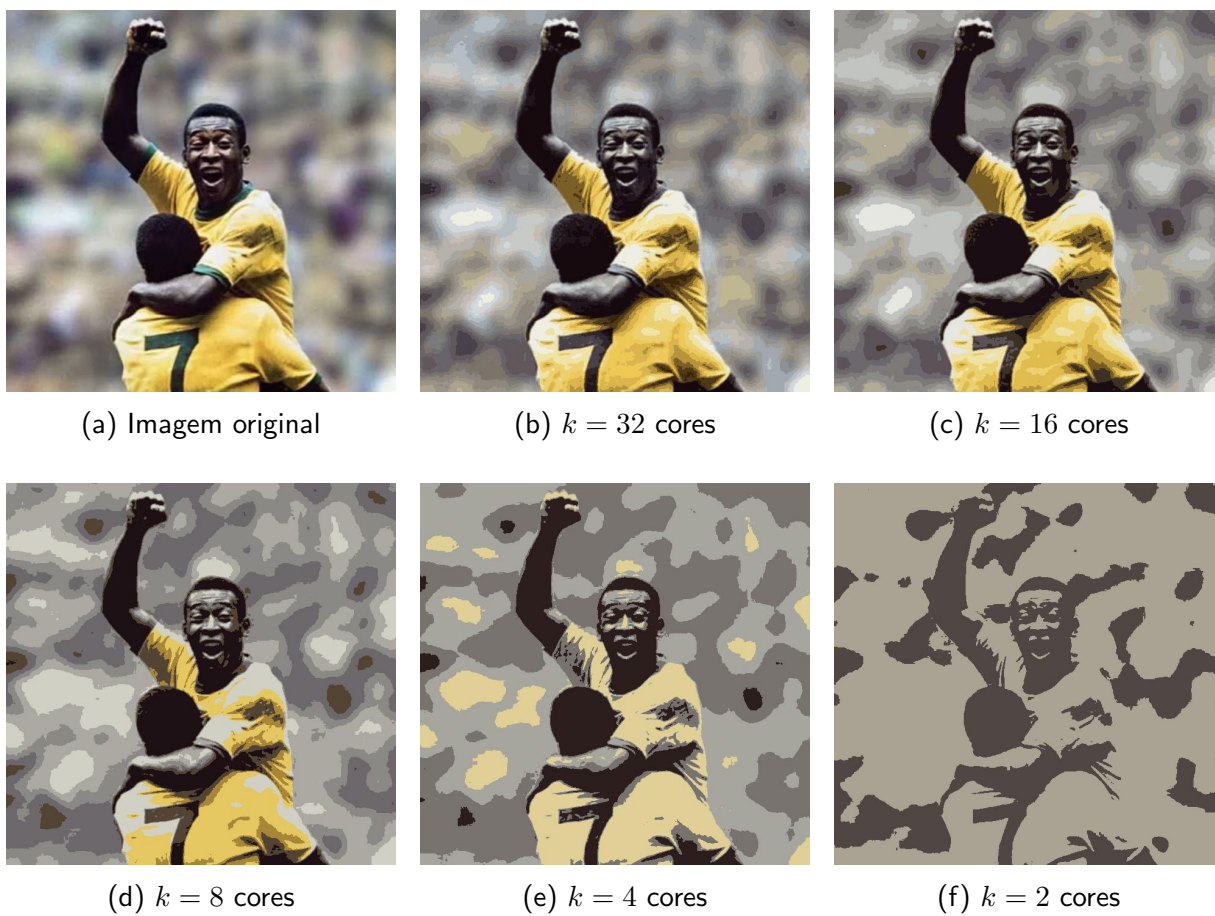
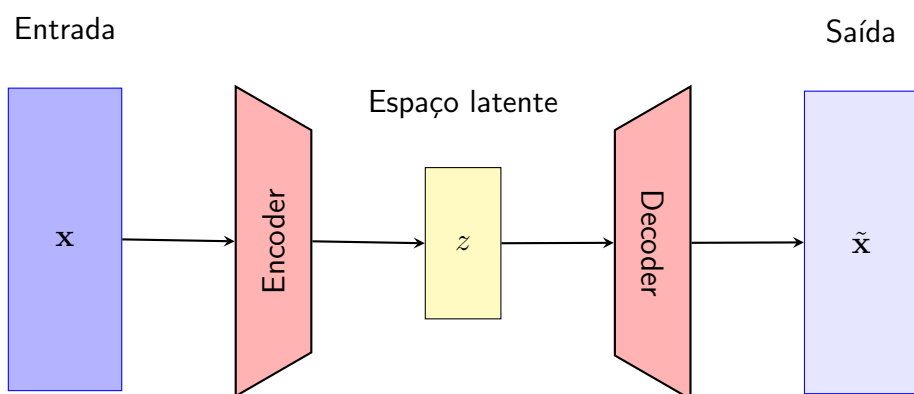
Figura 5: Quantização de cores usando k -means com diferentes valores de k 

Figura 6: Arquitetura de um autoencoder

3.1 Autoencoder Linear

O autoencoder linear é a versão mais simples de autoencoder. Ele usa uma rede neural com apenas uma camada oculta e uma função de ativação linear.

Matematicamente, ele reconstrói a entrada \mathbf{x} como:

$$\tilde{\mathbf{x}} = \mathbf{U}\mathbf{V}\mathbf{x}$$

em que $\mathbf{V} \in \mathbb{R}^{K \times D}$ é uma matriz de pesos que comprime os dados para o subespaço K -dimensional S e $\mathbf{U} \in \mathbb{R}^{D \times K}$ reconstrói a aproximação no espaço original.

Na solução ideal, \mathbf{U} e \mathbf{V} formam uma projeção ortogonal:

$$\mathbf{U} = \mathbf{Q}, \quad \mathbf{V} = \mathbf{Q}^\top \quad (2)$$

com \mathbf{Q} sendo uma base ortonormal que satisfaz $\mathbf{Q}^\top \mathbf{Q} = \mathbf{I}_K$.

Este modelo resolve simultaneamente dois problemas:

1. Minimiza o erro de reconstrução:

$$\arg\min_{\mathbf{U}, \mathbf{V}} \frac{1}{N} \sum_{i=1}^N \|\mathbf{x}^{(i)} - \tilde{\mathbf{x}}^{(i)}\|^2 \quad (3)$$

2. Maximiza a variância

$$\max \frac{1}{N} \sum_{i=1}^N \|\tilde{\mathbf{x}}^{(i)} - \boldsymbol{\mu}\|^2 \quad (4)$$

Geometricamente, o autoencoder linear projeta o vetor \mathbf{x} ortogonalmente em um subespaço linear (Veja Figura7). A projeção \mathbf{x}_{proj} é o ponto mais próximo de \mathbf{x} nesse subespaço. A diferença $\mathbf{x} - \mathbf{x}_{\text{proj}}$ é o erro de projeção.

A decomposição é dada por:

$$\mathbf{x} = \mathbf{x}_{\text{proj}} + (\mathbf{x} - \mathbf{x}_{\text{proj}})$$

em que \mathbf{x}_{proj} representa a parte explicada pelo modelo e $\mathbf{x} - \mathbf{x}_{\text{proj}}$ é o erro não explicado.

Pelo Teorema de Pitágoras:

$$\|\mathbf{x}\|^2 = \|\mathbf{x}_{\text{proj}}\|^2 + \|\mathbf{x} - \mathbf{x}_{\text{proj}}\|^2$$

Esta equação mostra que a energia (variância) total do vetor é a soma da energia (variância) explicada e do erro residual.

A solução deste problema é idêntica à Análise de Componentes Principais (PCA), que pode ser resolvida analiticamente usando projeções ortogonais.

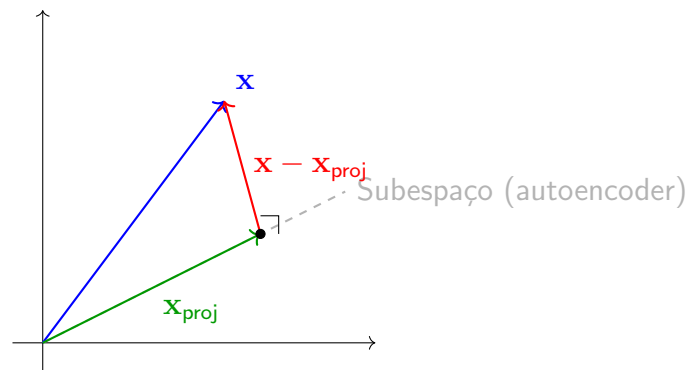


Figura 7: Projeção ortogonal no subespaço do autoencoder linear

A Figura 8 mostra a evolução da função de perda durante o treinamento do modelo de reconstrução linear. O valor da função de perda diminui consistentemente a cada época, indicando a convergência do algoritmo de otimização.

A Figura 9 apresenta os resultados da reconstrução para cinco dígitos manuscritos do conjunto de dados MNIST, utilizando um espaço latente de dimensão 64 (ou seja, os 400 pixels originais - 20×20 - são comprimidos em um vetor de 64 dimensões). A primeira linha exibe as imagens originais, enquanto a segunda linha mostra as reconstruções geradas pelo modelo. Os resultados demonstram uma reconstrução razoavelmente fiel, porém com um leve efeito de borramento, característico da compressão linear.

3.2 Autoencoder Convolucional

Os autoencoders não lineares superam as limitações dos modelos lineares ao aprender representações complexas em variedades não lineares (*manifolds*). Nesses modelos, o decodificador define o *manifold* como sua imagem, realizando uma redução de dimensionalidade que preserva estruturas não lineares nos dados.

Um exemplo de autoencoder não linear é a arquitetura convolucional a seguir que mapeia os dados para um espaço latente de 128 dimensões.

Arquitetura Convolucional

- **Codificador:**
 - 3 camadas convolucionais (filtros 3×3)
 - Ativação ReLU após cada convolução
 - *MaxPooling* para redução dimensional

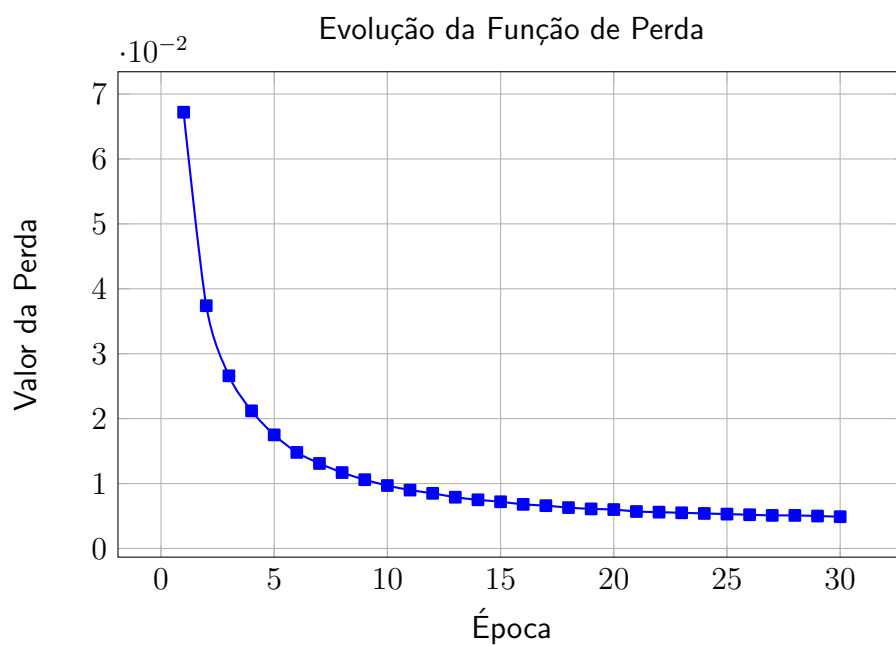


Figura 8: Evolução da função de perda durante o treinamento do autoencoder linear



Figura 9: Resultados de reconstrução do autoencoder linear para dígitos MNIST (acima: originais; abaixo: reconstruções)

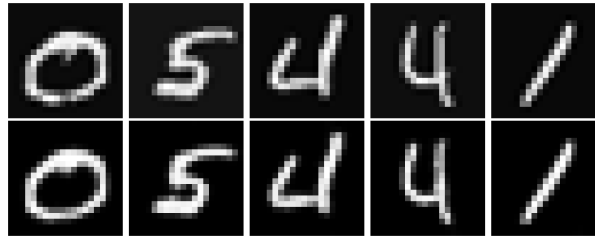


Figura 10: Resultados de reconstrução do autoencoder convolucional para dígitos MNIST (acima: originais; abaixo: reconstruções)

- Camada densa final (espaço latente de 128 dimensões)
- **Decodificador:**
 - Camada densa inicial para expansão
 - 3 camadas de convolução transposta
 - Função sigmoide na saída (normaliza para $[0, 1]$)

A Figura 10 apresenta os resultados da reconstrução com um autoencoder convolucional para cinco dígitos manuscritos do conjunto de dados MNIST no espaço latente de dimensão 128. A primeira linha exibe as imagens originais, enquanto a segunda linha mostra as reconstruções geradas pelo modelo. Os resultados demonstram alta fidelidade na reconstrução dos padrões e ausência de borramento.

A Figura 11 mostra a evolução da função de perda durante o treinamento do modelo de reconstrução linear. Note uma rápida diminuição nos primeiros passos, seguida por uma convergência gradual.

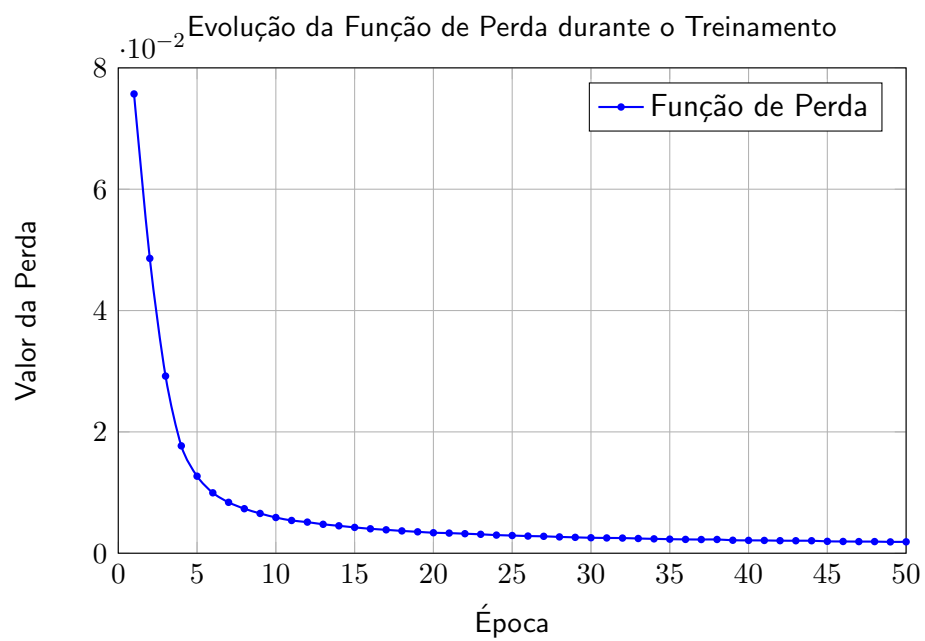


Figura 11: Evolução da função de perda do autoencoder convolucional.