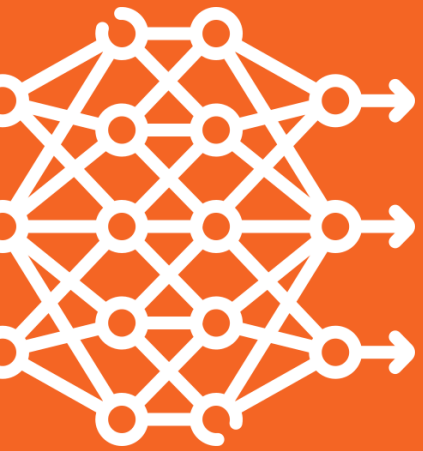


Classificação de dados gerados por aprendizado não supervisionado com Aprendizado Máquina Extremo

Eduardo Fonseca Rabelo - 11272697
Prof. Dr.: Odemir Martinez Bruno

Machine Learning

Vantagens:

- Otimização de **tempo;**
- **Customização** de produtos;
- Solução de **problemas.**

Machine Learning

Supervisionado e Não Supervisionado

Labeled and Unlabeled Data

Combinar entradas para produzir **previsões** úteis.

ML **Supervisionado** é a utilização de dados **rotulados** no treinamento.

Labels — Aquilo que queremos prever;

Feature — Variáveis de entrada;

Supervisionado

{feature, label}: (x,y)

Não supervisionado

{feature, label}: (x,?)

Labeled and Unlabeled Data

Combinar entradas para produzir **previsões** úteis.

ML **Supervisionado** é a utilização de dados **rotulados** no treinamento.

Labels — Aquilo que queremos prever;

Feature — Variáveis de entrada;

Supervisionado	Não supervisionado
{feature, label}: (x,y)	{feature, label}: (x,?)

\+ semi supervisionado + por reforço + ...

Supervisionado

{feature, label}: (x,y)

Não supervisionado

{feature, label}: (x,?)

- Redes Neurais Artificiais;
- Árvores de decisão;
- Máquinas de Vetores de Suporte.

Supervisionado

{feature, label}: (x,y)

- Redes Neurais Artificiais;
- Árvores de decisão;
- Máquinas de Vetores de Suporte.

Não supervisionado

{feature, label}: (x,?)

- Autoencoders;
- Clustering;
- PCA.

Autoencoders

Autoencoders

- Método não supervisionado;
- Extrair features removendo redundâncias*;

Autoencoders

- Método não supervisionado
- Extrair features removendo redundâncias*.

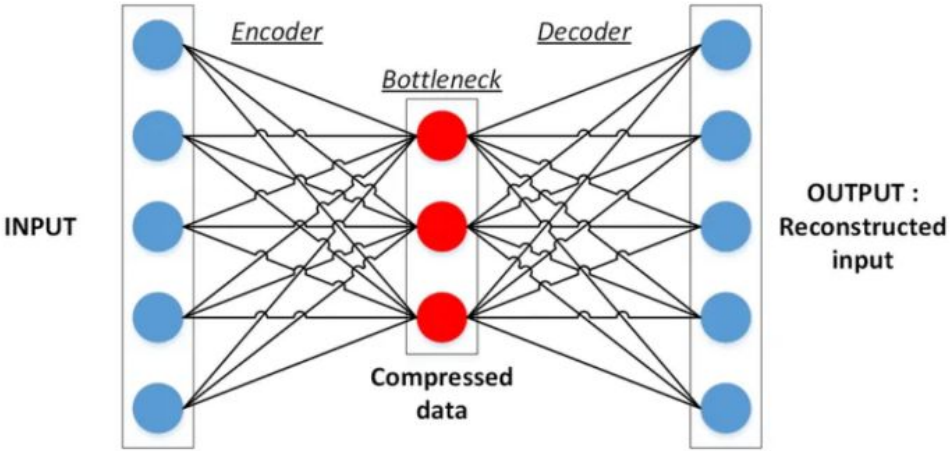
- Informações repetitivas

Autoencoders

- Método não supervisionado;
- Extrair features removendo redundâncias* - Informações repetitivas
- Remoção de ruídos;
- Compressão de dados (JPEG).

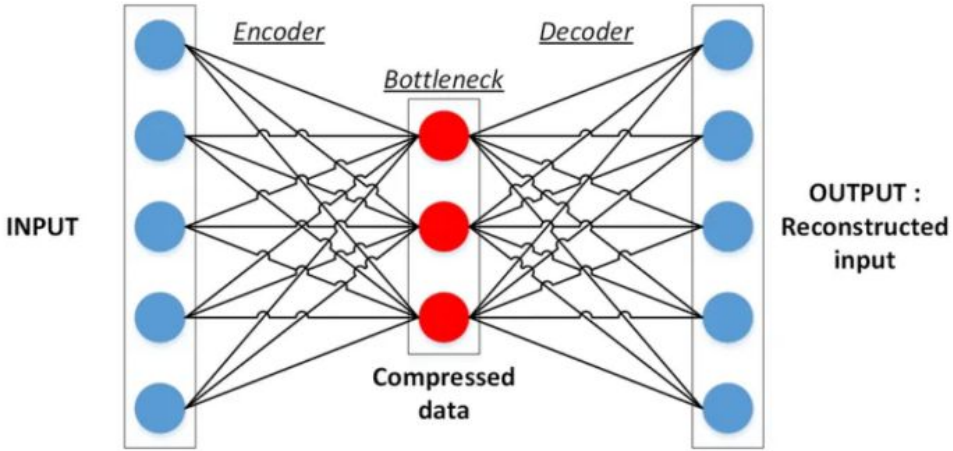
Autoencoders

Arquitetura básica:



Autoencoders

Arquitetura básica:

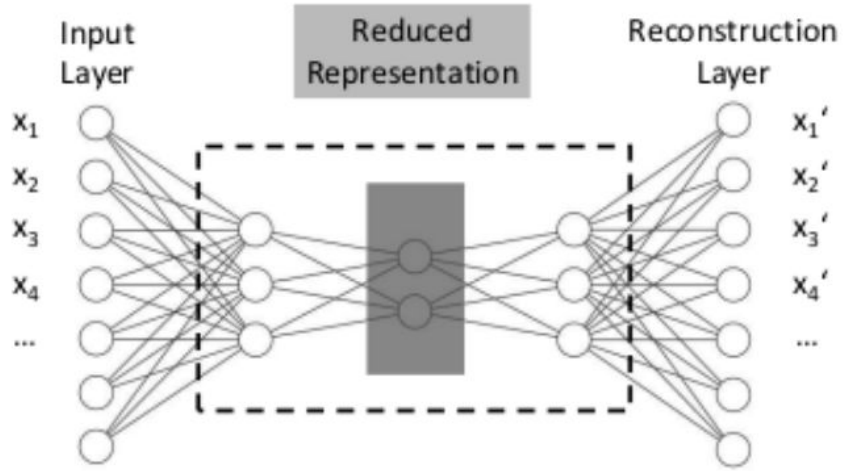


$$\mathbf{h} = g(\mathbf{x}) \quad \Longrightarrow \quad \mathbf{x'} = f(\mathbf{h})$$

$$\mathcal{L} = (\mathbf{x}, f(g(\mathbf{x})) = \mathcal{L}(\mathbf{x}, \mathbf{x'})$$

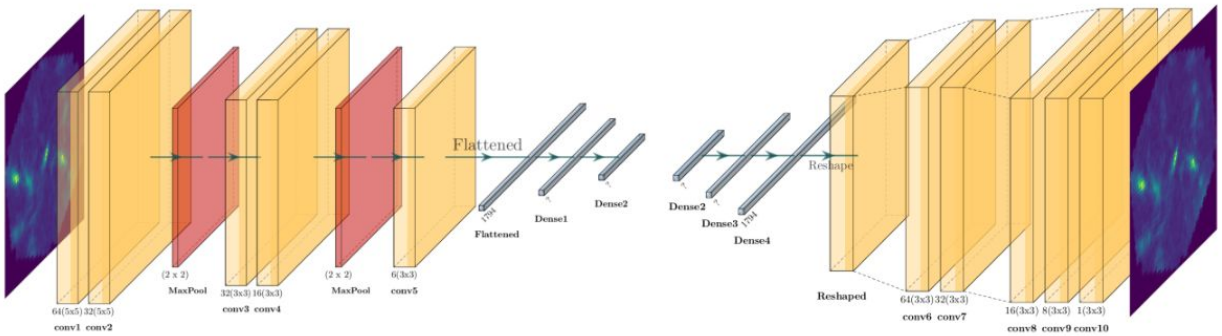
Autoencoders

Autoencoder profundo:



Autoencoders

Autoencoder convolucional:



(a) Encoder (b) Decoder

Extreme Learning Machine (ELM)

ELM

- Rede Feed-Forward;
- Apenas uma hidden layer;
- Treinamento pelo método de matriz pseudo-inversa;
- Não precisa de treinamento na input layer.

ELM

- Rede Feed-Forward;
- Apenas uma hidden layer;
- Treinamento pelo método de matriz pseudo-inversa;
- Não precisa de treinamento na input layer.

Entrada: $X = [x_1, x_2, \dots x_N], \quad x \in \mathbb{R}$

Saída: $$Y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} y_{11} & \cdots & y_{1D} \\ \vdots & \ddots & \vdots \\ y_{N1} & \cdots & y_{ND} \end{bmatrix}$$

ELM

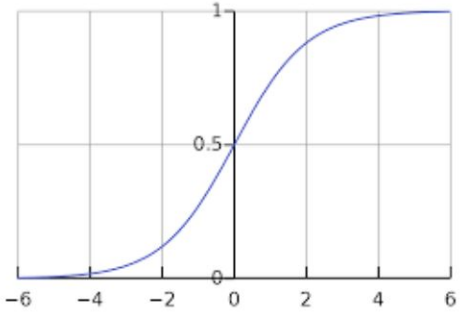
Entrada: $X = [x_1, x_2, \dots x_N], \quad x \in \mathbb{R}$

Saída: $Y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} y_{11} & \cdots & y_{1D} \\ \vdots & \ddots & \vdots \\ y_{N1} & \cdots & y_{ND} \end{bmatrix}$

Função de ativação:

$$h(x) = [h_1, h_2, \dots, h_L], \quad h(x) = g(x, w, c)$$

$$g(x, w, c) = \frac{1}{1 + \exp^{-(wx+c)}}$$



ELM

Entrada: $X = [x_1, x_2, \dots x_N], \quad x \in \mathbb{R}$

Saída: $$Y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} y_{11} & \cdots & y_{1D} \\ \vdots & \ddots & \vdots \\ y_{N1} & \cdots & y_{ND} \end{bmatrix}$$

$$H = \begin{bmatrix} h(x_1) \\ \vdots \\ h(x_N) \end{bmatrix} = \begin{bmatrix} h_1(x_1) & \cdots & h_L(x_1) \\ \vdots & \ddots & \vdots \\ h_1(x_N) & \cdots & h_L(x_N) \end{bmatrix}$$

$$H = \begin{bmatrix} g(w_1 \cdot x_1 + c_1) & \cdots & g(w_L \cdot x_1 + c_L) \\ \vdots & \ddots & \vdots \\ g(w_1 \cdot x_N + c_1) & \cdots & g(w_L \cdot x_N + c_L) \end{bmatrix}_{N \times L}$$

Valores aleatórios

$$h(x) = [h_1, h_2, \dots, h_L], \quad h(x) = g(x, w, c)$$

Função de ativação:

$$g(x, w, c) = \frac{1}{1 + \exp^{-(wx+c)}}$$

$$w = [w_1, w_2, \dots, w_L]$$

$$c = [c_1, c_2, \dots, c_L]$$

ELM

Entrada: $X = [x_1, x_2, \dots x_N], \quad x \in \mathbb{R}$

Saída: $Y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} y_{11} & \cdots & y_{1D} \\ \vdots & \ddots & \vdots \\ y_{N1} & \cdots & y_{ND} \end{bmatrix}$

$$H = \begin{bmatrix} h(x_1) \\ \vdots \\ h(x_N) \end{bmatrix} = \begin{bmatrix} h_1(x_1) & \cdots & h_L(x_1) \\ \vdots & \ddots & \vdots \\ h_1(x_N) & \cdots & h_L(x_N) \end{bmatrix}$$

$$H = \begin{bmatrix} g(w_1 \cdot x_1 + c_1) & \cdots & g(w_L \cdot x_1 + c_L) \\ \vdots & \ddots & \vdots \\ g(w_1 \cdot x_N + c_1) & \cdots & g(w_L \cdot x_N + c_L) \end{bmatrix}_{N \times L}$$

Valores aleatórios

$$h(x) = [h_1, h_2, \dots, h_L], \quad h(x) = g(x, w, c)$$

Função de ativação:

$$g(x, w, c) = \frac{1}{1 + \exp^{-(wx+c)}}$$

$$w = [w_1, w_2, \dots, w_L]$$

$$c = [c_1, c_2, \dots, c_L]$$

ELM

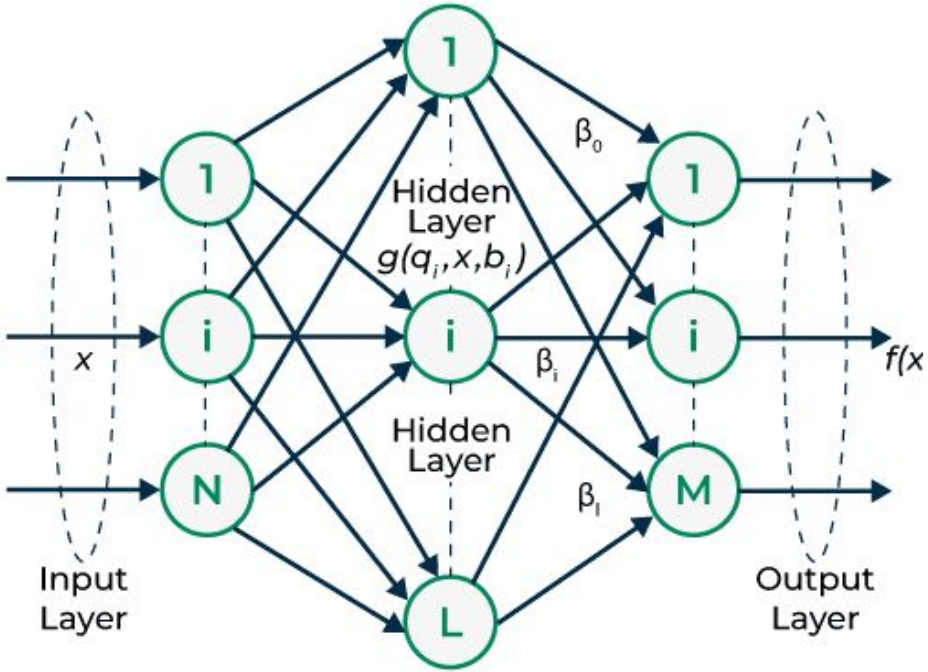
Pseudo-inversa

$$\beta = H^{\dagger}Y$$

Contém os pesos recalculados. ⟶ $\beta = [\beta_1, \beta_2, \dots, \beta_L]^T$

$$\beta = (H^T H + \frac{I}{C})^{-1} H^T Y$$

ELM


1
1
i
N
x
Input
Layer
Hidden
Layer
$g(q_i,x,b_i)$
i
Hidden
Layer
L
β_0
β_i
β_l
1
i
M
f(x
Output
Layer

Métodos e Resultados

Banco de dados - MNIST

- 60000 imagens de teste e 10000 imagens de treino;
- Imagens 28x28 pixels.

```
#Importação dos features/labels de treino e features/labels de teste do MNIST
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Normalizar os dados
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Adicionar uma dimensão de canal
x_train = np.expand_dims(x_train, axis=-1)
x_test = np.expand_dims(x_test, axis=-1)
```

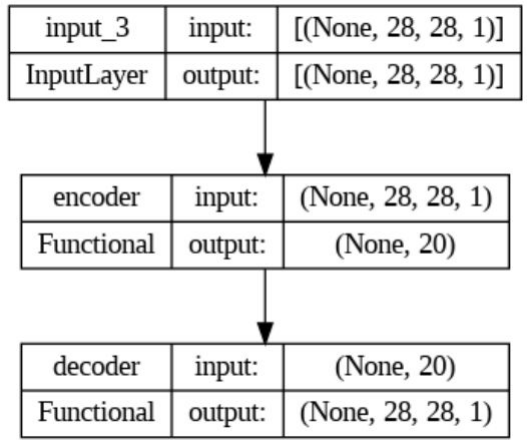
Banco de dados - MNIST

- 60000 imagens de teste e 10000 imagens de treino;
- Imagens 28x28 pixels.

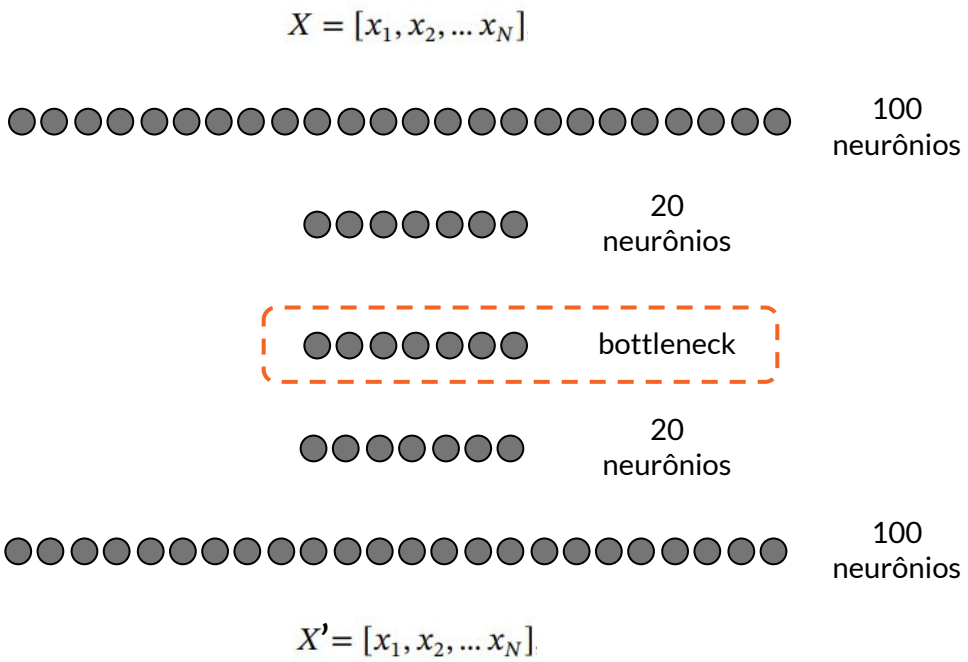
Autoencoder profundo:

```
x_train_shape = (28, 28, 1)

# Encoder
hidden_size = 100
latent_size = 20

input_layer = layers.Input(shape=x_train_shape)
flattened = layers.Flatten()(input_layer)
hidden = layers.Dense(hidden_size, activation='relu')(flattened)
latent = layers.Dense(latent_size, activation='relu')(hidden)
encoder = Model(inputs=input_layer, outputs=latent, name='encoder')
encoder.summary()

# Decoder
input_layer_decoder = layers.Input(shape=(latent_size,))
upsampled = layers.Dense(hidden_size, activation='relu')(input_layer_decoder)
upsampled = layers.Dense(encoder.layers[1].output_shape[-1], activation='relu')(upsampled)
constructed = layers.Reshape(x_train_shape)(upsampled)
decoder = Model(inputs=input_layer_decoder, outputs=constructed, name='decoder')
decoder.summary()

# Autoencoder (combinação do encoder e decoder)
autoencoder_input = layers.Input(shape=x_train_shape)
encoded_img = encoder(autoencoder_input)
decoded_img = decoder(encoded_img)
autoencoder = Model(inputs=autoencoder_input, outputs=decoded_img, name='autoencoder')
autoencoder.summary()
```

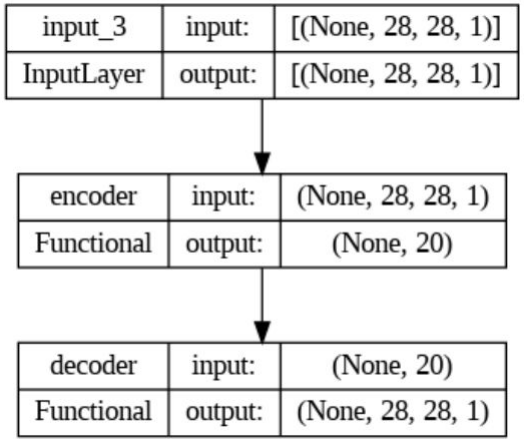
Arquitetura



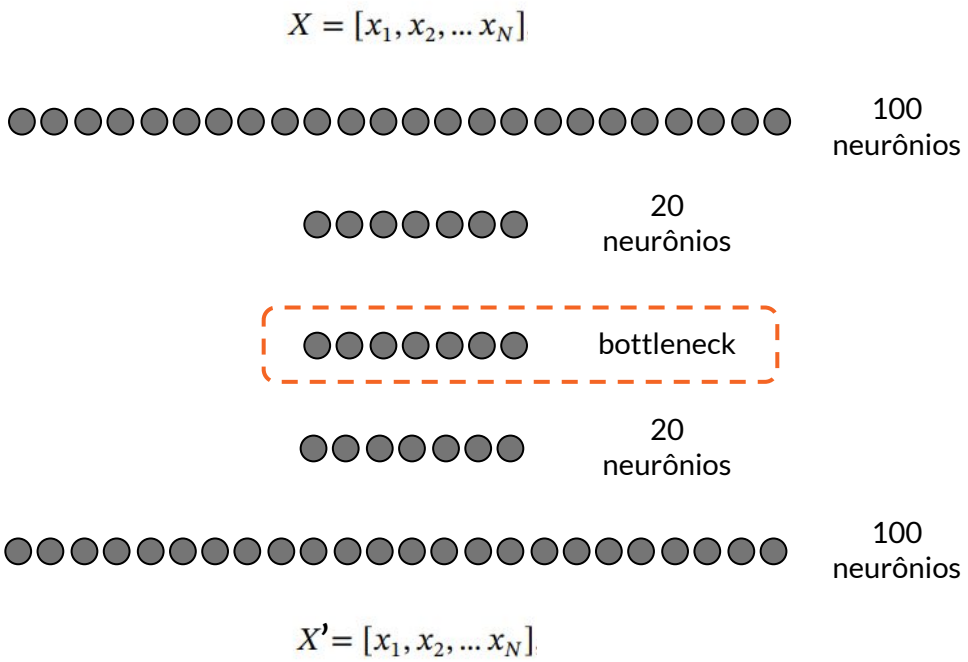
Autoencoder profundo:



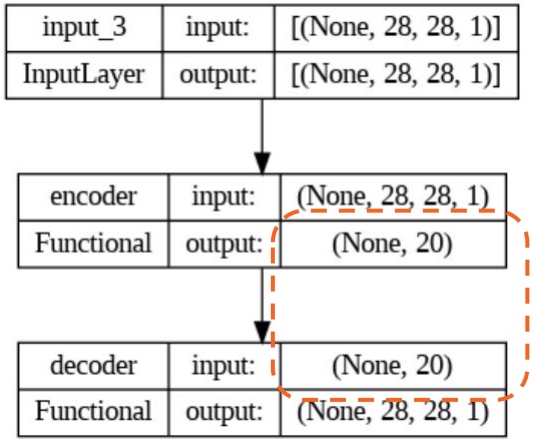
Arquitetura



Autoencoder profundo:

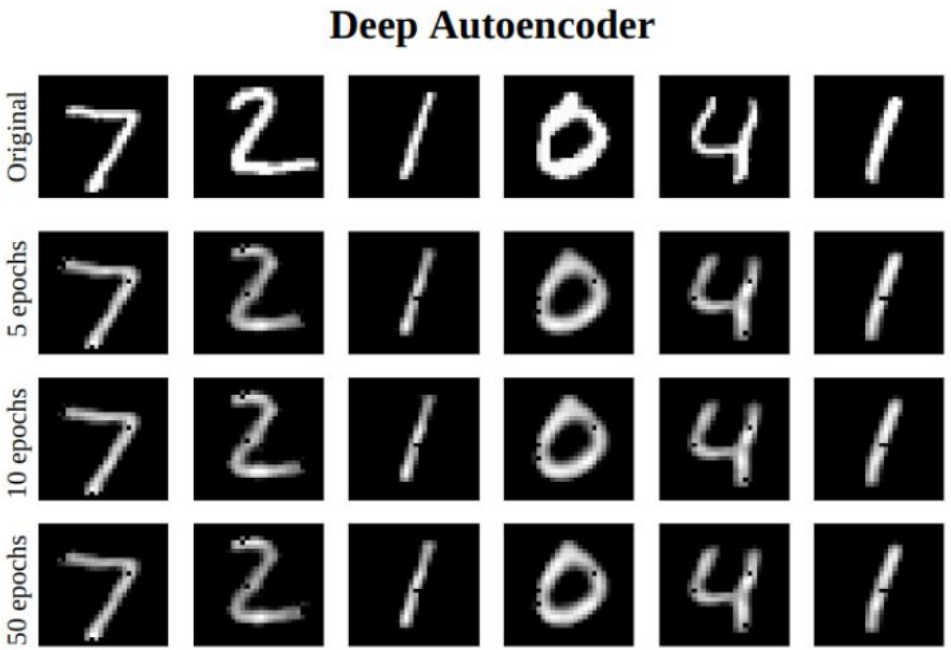


Arquitetura



Autoencoder profundo:

- O número de epochs não pareceu ser muito importante nesse caso.



Autoencoder profundo:

- Imagens reconstruídas não parecem ter diferença mas...

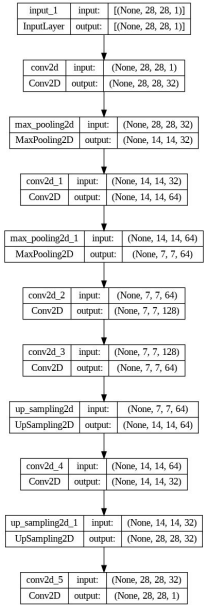
Autoencoder profundo:

- Imagens reconstruídas não parecem ter diferença mas...

Autoencoder convolucional:

```
# Codificador (Encoder)
encoder_input = layers.Input(shape=input_shape)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(encoder_input)
x = layers.MaxPooling2D((2, 2), padding='same')(x)  # 14x14x32
x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = layers.MaxPooling2D((2, 2), padding='same')(x)  # 7x7x64
encoder_output = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)

encoder = models.Model(encoder_input, encoder_output)
encoder.summary()

# Decodificador (Decoder)
x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(encoder_output)
x = layers.UpSampling2D((2, 2))(x)  # 14x14x64
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)  # 28x28x32
decoder_output = layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

decoded = models.Model(encoder_output, decoder_output)

autoencoder = models.Model(encoder_input, decoder_output)
autoencoder.summary()
```

Arquitetura



Autoencoder convolucional:

```python
# Codificador (Encoder)
encoder_input = layers.Input(shape=input_shape)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(encoder_input)
x = layers.MaxPooling2D((2, 2), padding='same')(x)  # 14x14x32
x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = layers.MaxPooling2D((2, 2), padding='same')(x)  # 7x7x64
encoder_output = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)

encoder = models.Model(encoder_input, encoder_output)
encoder.summary()

# Decodificador (Decoder)
x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(encoder_output)
x = layers.UpSampling2D((2, 2))(x)  # 14x14x64
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)  # 28x28x32
decoder_output = layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

decoded = models.Model(encoder_output, decoder_output)

autoencoder = models.Model(encoder_input, decoder_output)
autoencoder.summary()
```

Arquitetura

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 28, 28, 1)]	0
conv2d_6 (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_7 (Conv2D)	(None, 14, 14, 64)	18496
max_pooling2d_3 (MaxPooling2D)	(None, 7, 7, 64)	0
conv2d_8 (Conv2D)	(None, 7, 7, 128)	73856
conv2d_9 (Conv2D)	(None, 7, 7, 64)	73792
up_sampling2d_2 (UpSampling2D)	(None, 14, 14, 64)	0
conv2d_10 (Conv2D)	(None, 14, 14, 32)	18464
up_sampling2d_3 (UpSampling2D)	(None, 28, 28, 32)	0
conv2d_11 (Conv2D)	(None, 28, 28, 1)	289

Total params: 185217 (723.50 KB)
Trainable params: 185217 (723.50 KB)
Non-trainable params: 0 (0.00 Byte)

Autoencoder convolucional:

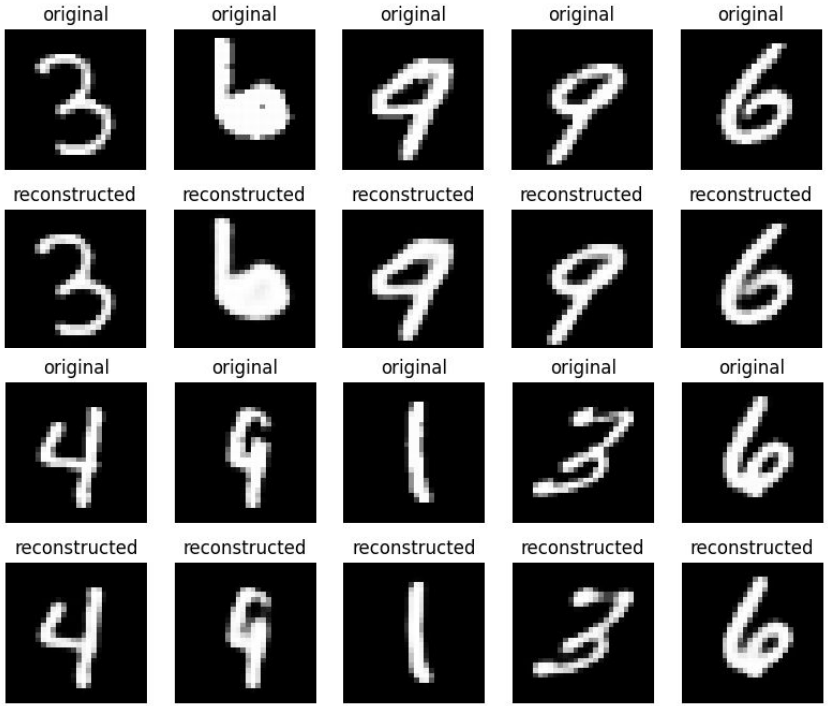
```python
# Codificador (Encoder)
encoder_input = layers.Input(shape=input_shape)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(encoder_input)
x = layers.MaxPooling2D((2, 2), padding='same')(x)  # 14x14x32
x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = layers.MaxPooling2D((2, 2), padding='same')(x)  # 7x7x64
encoder_output = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)

encoder = models.Model(encoder_input, encoder_output)
encoder.summary()

# Decodificador (Decoder)
x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(encoder_output)
x = layers.UpSampling2D((2, 2))(x)  # 14x14x64
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)  # 28x28x32
decoder_output = layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

decoded = models.Model(encoder_output, decoder_output)

autoencoder = models.Model(encoder_input, decoder_output)
autoencoder.summary()
```

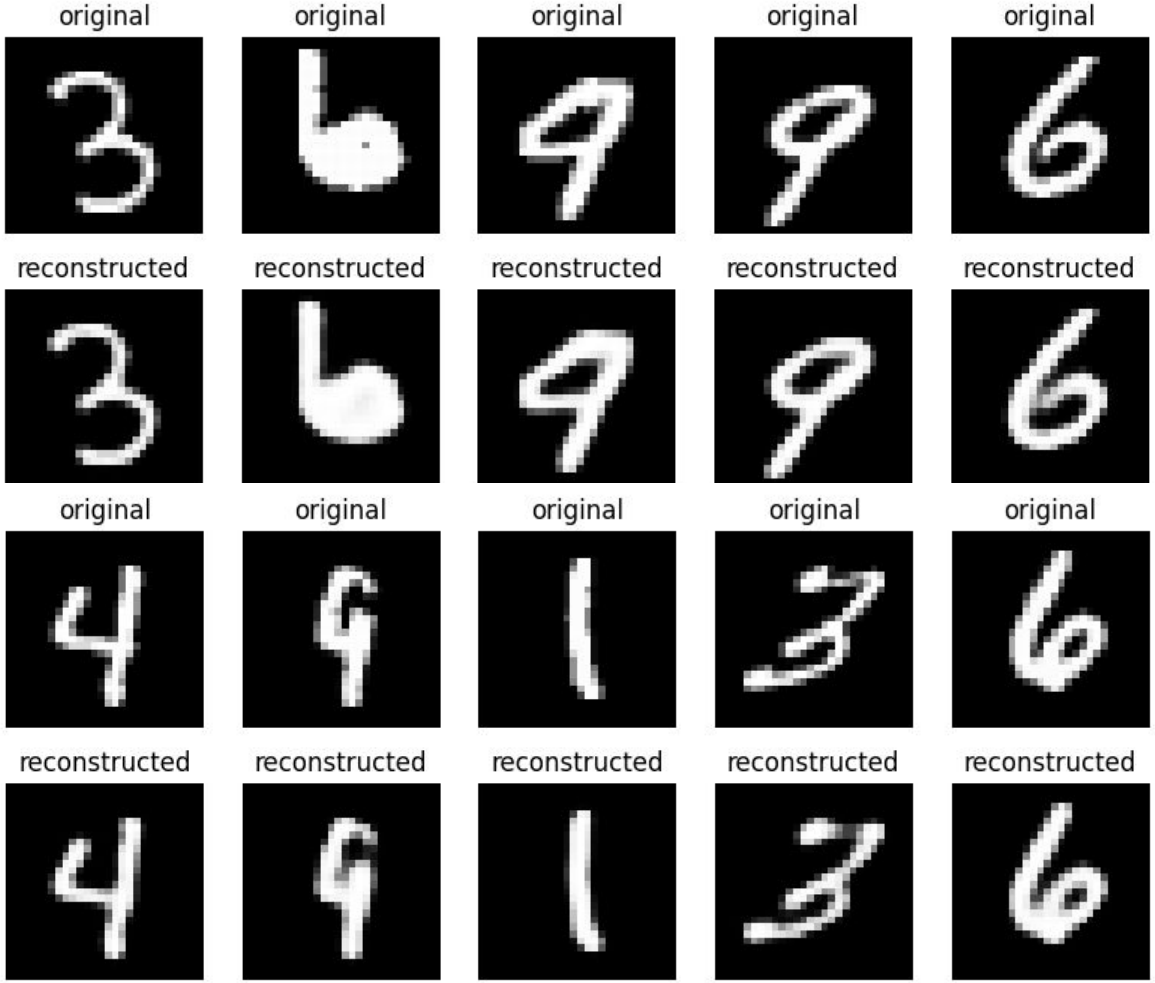
Arquitetura

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 28, 28, 1)]	0
conv2d_6 (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_7 (Conv2D)	(None, 14, 14, 64)	18496
max_pooling2d_3 (MaxPooling2D)	(None, 7, 7, 64)	0
conv2d_8 (Conv2D)	(None, 7, 7, 128)	73856
conv2d_9 (Conv2D)	(None, 7, 7, 64)	73792
up_sampling2d_2 (UpSampling2D)	(None, 14, 14, 64)	0
conv2d_10 (Conv2D)	(None, 14, 14, 32)	18464
up_sampling2d_3 (UpSampling2D)	(None, 28, 28, 32)	0
conv2d_11 (Conv2D)	(None, 28, 28, 1)	289

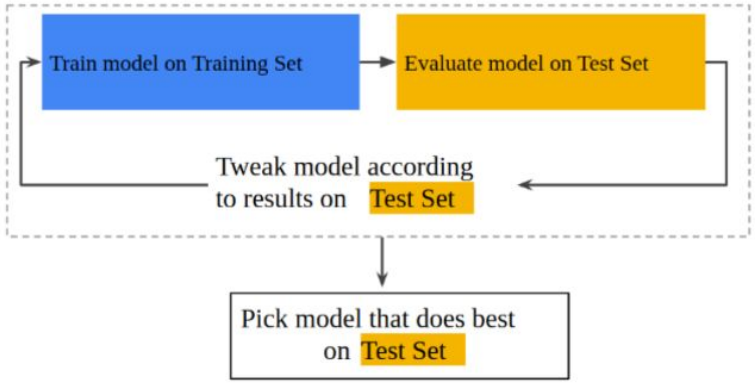
Total params: 185217 (723.50 KB)
Trainable params: 185217 (723.50 KB)
Non-trainable params: 0 (0.00 Byte)

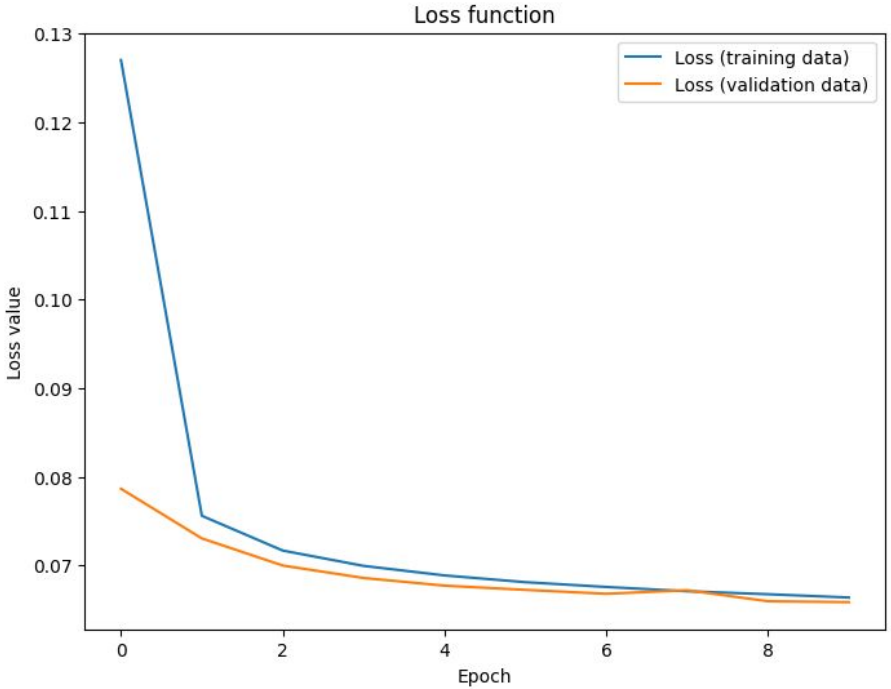
Autoencoder convolucional:




original
original
original
original
original
reconstructed
reconstructed
reconstructed
reconstructed
reconstructed
original
original
original
original
original
reconstructed
reconstructed
reconstructed
reconstructed
reconstructed

Autoencoder convolucional:





ELM

```
# Carregar e preprocessar o dataset MNIST
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(x_train.shape[0], -1).astype('float32') / 255.0
x_test = x_test.reshape(x_test.shape[0], -1).astype('float32') / 255.0

# One-hot encoding das labels
encoder = OneHotEncoder(sparse=False)
y_train_onehot = encoder.fit_transform(y_train.reshape(-1, 1))
y_test_onehot = encoder.transform(y_test.reshape(-1, 1))
```

Converte as labels categóricas em uma representação numérica

ELM

```
# Carregar e preprocessar o dataset MNIST
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(x_train.shape[0], -1).astype('float32') / 255.0
x_test = x_test.reshape(x_test.shape[0], -1).astype('float32') / 255.0

# One-hot encoding das labels
encoder = OneHotEncoder(sparse=False)
y_train_onehot = encoder.fit_transform(y_train.reshape(-1, 1))
y_test_onehot = encoder.transform(y_test.reshape(-1, 1))
```

Converte as labels categóricas em uma representação numérica

```
# Parâmetros do modelo ELM
input_size = x_train.shape[1]
hidden_size = 1000  # número de neurônios na camada oculta
output_size = y_train_onehot.shape[1]
```

Definimos o tamanho do modelo

ELM

```
# Inicializar pesos da camada oculta aleatoriamente
W = np.random.randn(input_size, hidden_size)
b = np.random.randn(hidden_size)
```

Inicialização de pesos aleatórios

ELM

```
# Inicializar pesos da camada oculta aleatoriamente
W = np.random.randn(input_size, hidden_size)
b = np.random.randn(hidden_size)
```

Inicialização de pesos aleatórios

```
# Função de ativação da camada oculta (ReLU)
def relu(x):
    return np.maximum(0, x)

# Calcular a saída da camada oculta
H = relu(np.dot(x_train, W) + b)
```

Criação de matrizes de ativação

ELM

```
# Calcular os pesos da camada de saída usando pseudoinversa
H_pinv = pinv(H)
beta = np.dot(H_pinv, y_train_onehot)

# Fazer previsões no conjunto de teste
H_test = relu(np.dot(x_test, W) + b)
y_pred = np.dot(H_test, beta)

# Converter as previsões para labels
y_pred_labels = np.argmax(y_pred, axis=1)

# Avaliar a acurácia
accuracy = np.mean(y_pred_labels == y_test)
print(f'Test accuracy: {accuracy * 100:.2f}%')
```

Cálculo dos pesos da matriz beta pela pseudo-inversa

ELM

```
# Calcular os pesos da camada de saída usando pseudoinversa
H_pinv = pinv(H)
beta = np.dot(H_pinv, y_train_onehot)

# Fazer previsões no conjunto de teste
H_test = relu(np.dot(x_test, W) + b)
y_pred = np.dot(H_test, beta)

# Converter as previsões para labels
y_pred_labels = np.argmax(y_pred, axis=1)

# Avaliar a acurácia
accuracy = np.mean(y_pred_labels == y_test)
print(f'Test accuracy: {accuracy * 100:.2f}%')
```

Cálculo dos pesos da matriz beta pela pseudo-inversa

Teste de dados

ELM

```
# Calcular os pesos da camada de saída usando pseudoinversa
H_pinv = pinv(H)
beta = np.dot(H_pinv, y_train_onehot)

# Fazer previsões no conjunto de teste
H_test = relu(np.dot(x_test, W) + b)
y_pred = np.dot(H_test, beta)

# Converter as previsões para labels
y_pred_labels = np.argmax(y_pred, axis=1)

# Avaliar a acurácia
accuracy = np.mean(y_pred_labels == y_test)
print(f'Test accuracy: {accuracy * 100:.2f}%')
```

Cálculo dos pesos da matriz beta pela pseudo-inversa

Teste de dados

Verificação de Acurácia

ELM

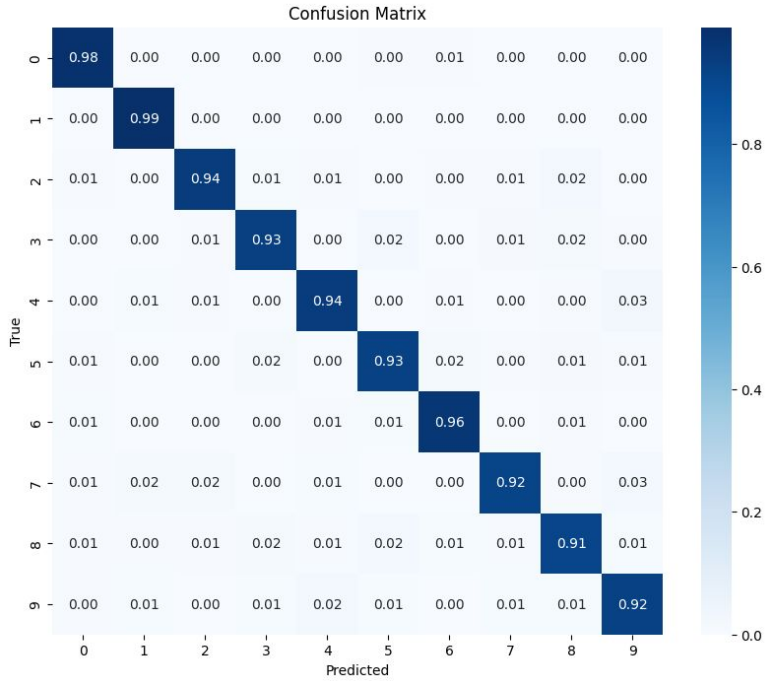
INPUT	ACURÁCIA
MNIST	
DEEP_AUTOENCODER	
CONV_AUTOENCODER	

ELM

INPUT	ACURÁCIA
MNIST	94,34%
DEEP_AUTOENCODER	
CONV_AUTOENCODER	

ELM

INPUT	ACURÁCIA
MNIST	94,34%
DEEP_AUTOENCODER	
CONV_AUTOENCODER	



ELM

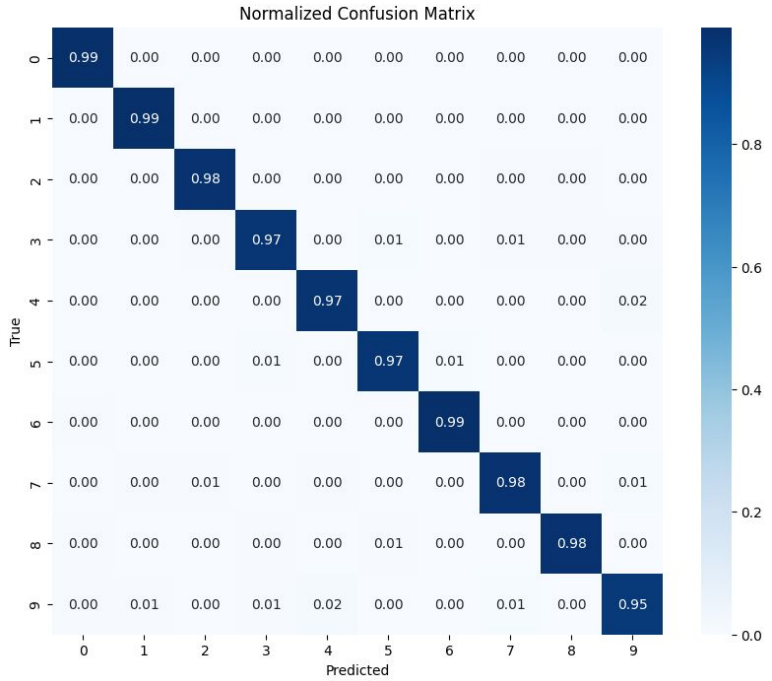
INPUT	ACURÁCIA
MNIST	94,34%
DEEP_AUTOENCODER	
CONV_AUTOENCODER	

ELM

INPUT	ACURÁCIA
MNIST	94,34%
DEEP_AUTOENCODER	97,65%
CONV_AUTOENCODER	

ELM

INPUT	ACURÁCIA
MNIST	94,34%
DEEP_AUTOENCODER	97,65%
CONV_AUTOENCODER	



ELM

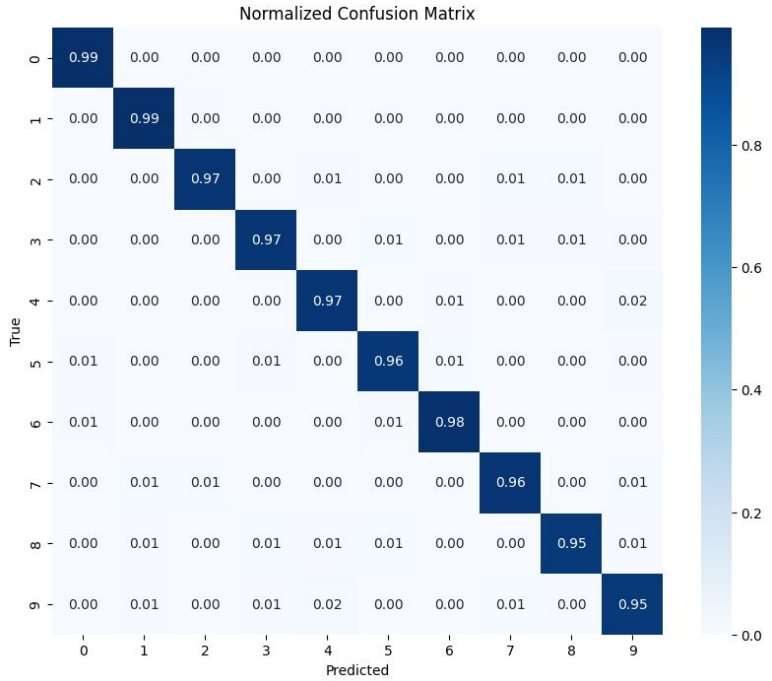
INPUT	ACURÁCIA
MNIST	94,34%
DEEP_AUTOENCODER	97,65%
CONV_AUTOENCODER	97,05%

ELM

INPUT	ACURÁCIA
MNIST	94,34%
DEEP_AUTOENCODER	97,65%
CONV_AUTOENCODER	97,05%

ELM

INPUT	ACURÁCIA
MNIST	94,34%
DEEP_AUTOENCODER	97,65%
CONV_AUTOENCODER	97,05%



Conclusões

Conclusões

Autoencoders como bons compactadores de imagem, guardando informação relevante apenas em 7 neurônios;

Em autoencoders convolucionais compactou uma imagem 28x28 para 7x7 e foi capaz de reconstruir sem muitos detalhes significativos;

ELM é um bom classificador uma vez que apresentou uma acurácia de 94, 98 e 97% nos dados utilizados.

Conclusões

Possibilidades de verificação:

Trabalhar com banco de dados mais detalhados;

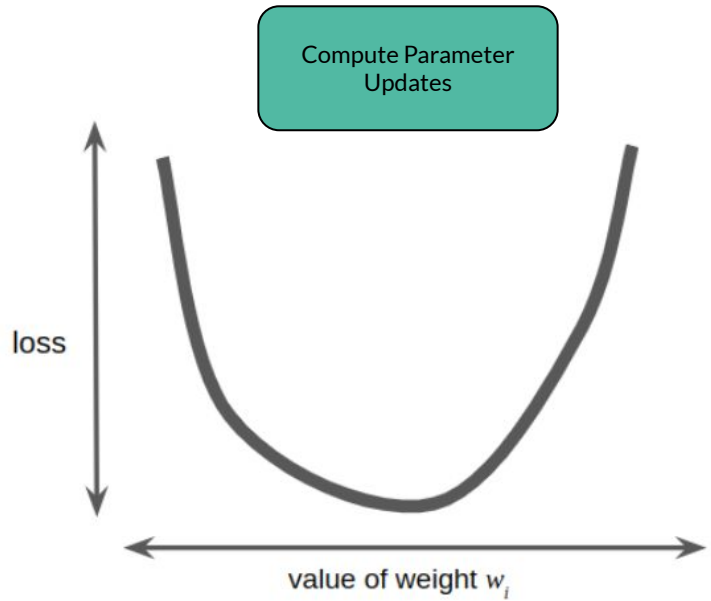
Comparação em métodos de compactação de dados;

Trabalhar com dados ruidosos.

Obrigado!

Gradient Descent

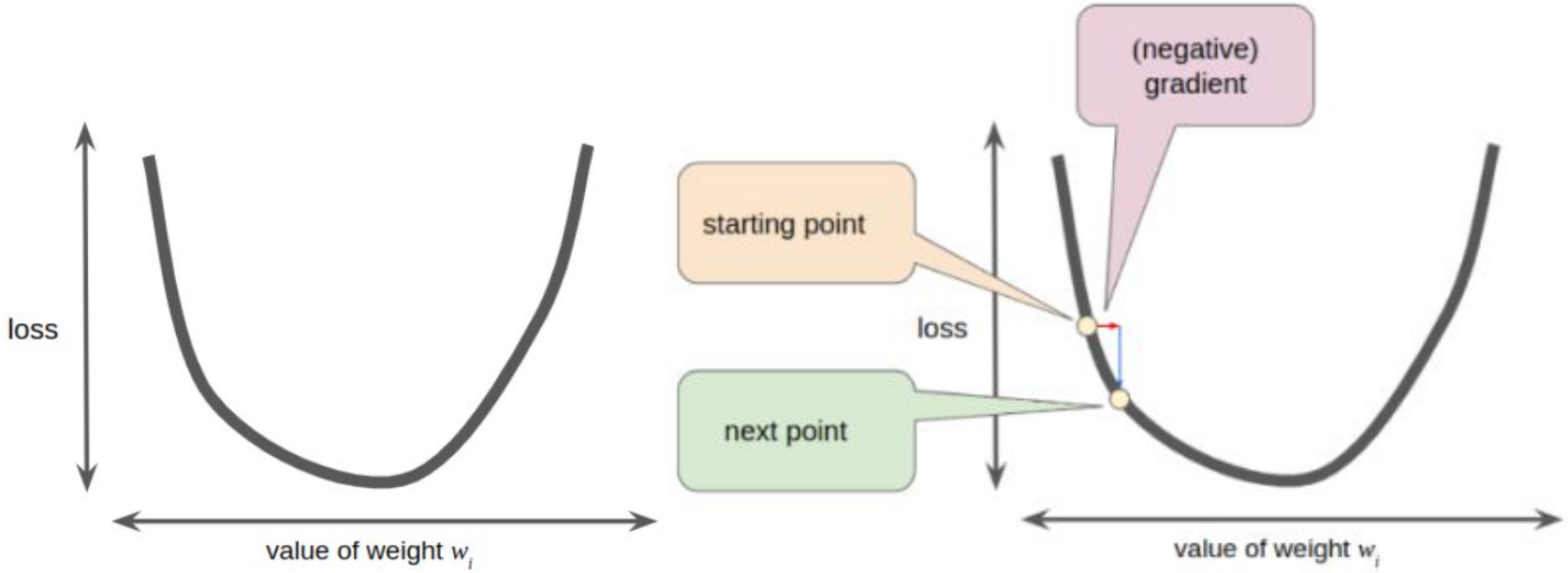
Mas qual o procedimento tomado em:



- Curva **convexa**;
- Apenas **um mínimo**;
- Análise completa complicada;

Gradient Descent

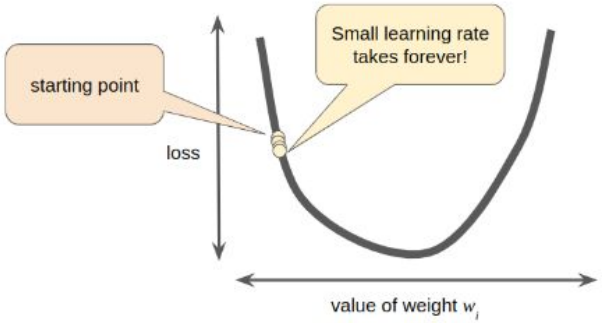
Estabelecer **pontos de partida** como w1 = 0 e tomar o **gradiente** do ponto.



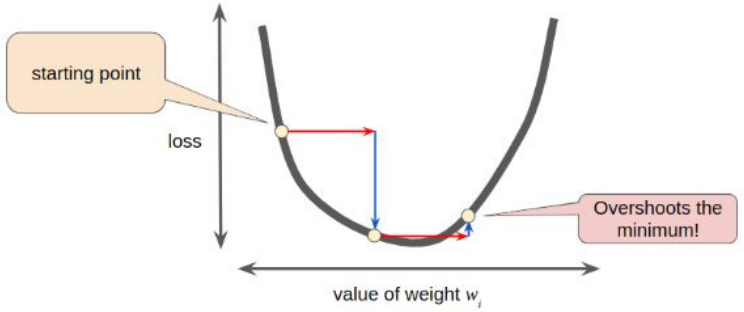
Gradient Descent: Parâmetros

Learning Rate ou **Step Size**: São hiperparâmetros ajustados pelo programador que ajustam o aprendizado.

Learning Rate **baixo**:



Learning Rate **alto**:



Redução de erros

Queremos trabalhar com modelos que não apresente um grau de complexidade muito alto.

Devemos então trabalhar com uma quantidade de dados razoável para o problema adotado para que possamos evitar trabalhar com um modelo que sempre apresenta **overfit** ou que apresenta uma **complexidade** maior que a necessária.

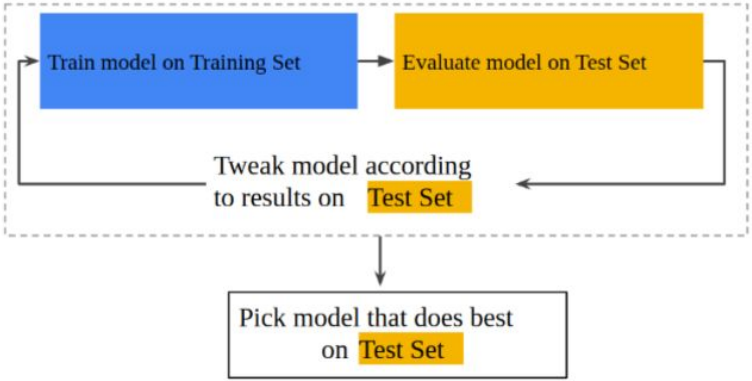
Redução de erros

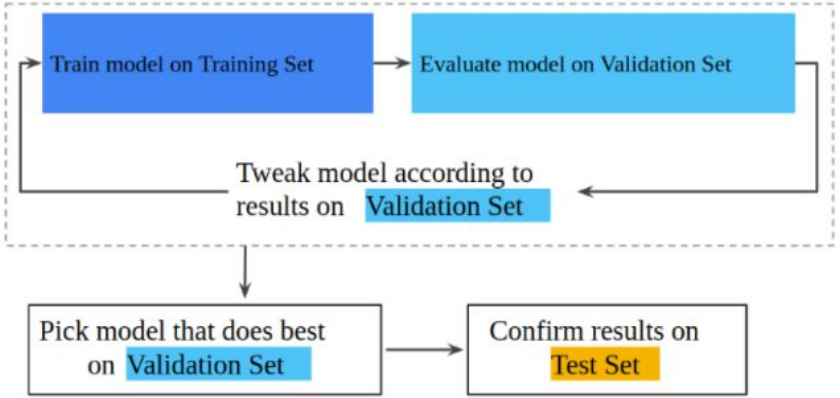
Portanto, podemos **separar** nosso conjunto de dados em um **subconjunto** que nos auxilie a **verificar** nossos resultados.

O **training set** deve ser grande o suficiente para ter algum s**ignificado estatístico** para o modelo com um **conjunto de dados variados**.

Redução de erros

Checagem dupla a partir de um novo subconjunto, o grupo de **validação**:





Redução de erros

Grupo de **validação**:

A validação é feita durante o **treinamento** do modelo para **ajudar** a escolher de forma **imparcial** os melhores parâmetros. Finalmente, o modelo de **teste** é usado **após o treinamento** para verificar a eficiência do aprendizado.

Dados sintéticos
Feature Crosses

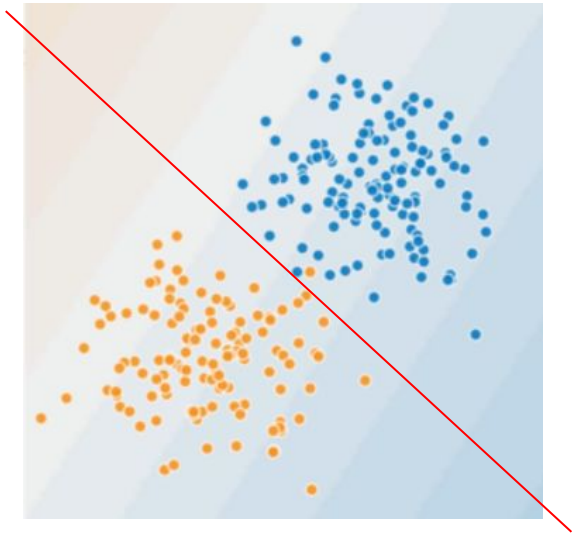
Linearidade e Não Linearidade

Os modelos de ML em certo momento podem começar a lidar com exemplos que não trazer a característica da **linearidade**:

$$y' = b + w_1 x_1 + w_2 x_2$$

Linearidade e Não Linearidade

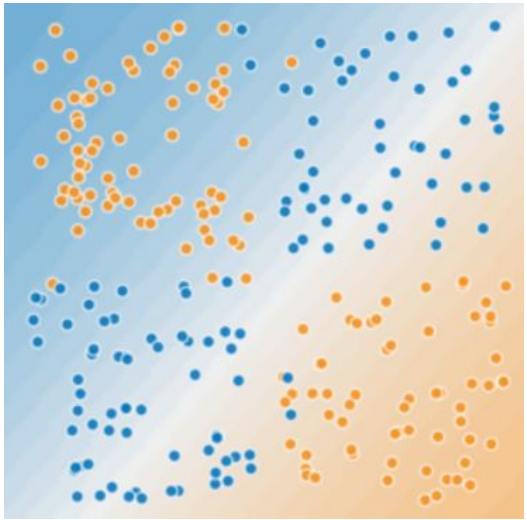
Os modelos de ML em certo momento podem começar a lidar com exemplos que não trazer a característica da **linearidade**:



$$y' = b + w_1 x_1 + w_2 x_2$$

Linearidade e Não Linearidade

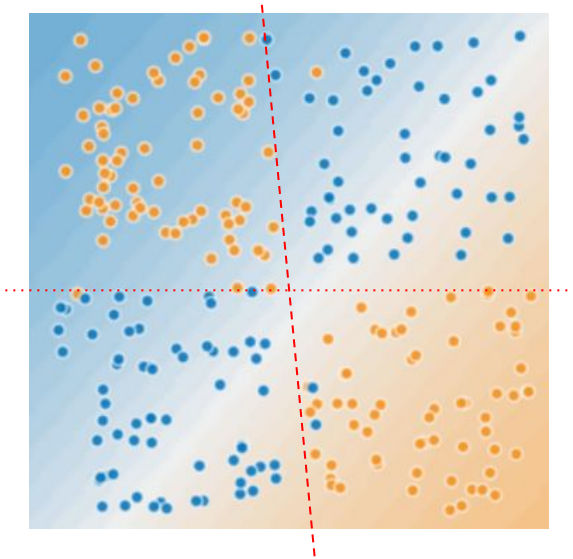
Os modelos de ML em certo momento podem começar a lidar com exemplos que não trazer a característica da **linearidade**:



$$y' = b + w_1 x_1 + w_2 x_2$$

Linearidade e Não Linearidade

Os modelos de ML em certo momento podem começar a lidar com exemplos que não trazer a característica da **linearidade**:



$$y' = b + w_1 x_1 + w_2 x_2$$

Uma reta não é suficiente para classificar o conjunto

Linearidade e Não Linearidade

Podemos criar um **feature sintético** a partir de dois features que apresentam características lineares.

- $x_3 = x_1 x_2$
- Introdução da **não linearidade** ao modelo;
- É visto como apenas mais um atributo pelo modelo:

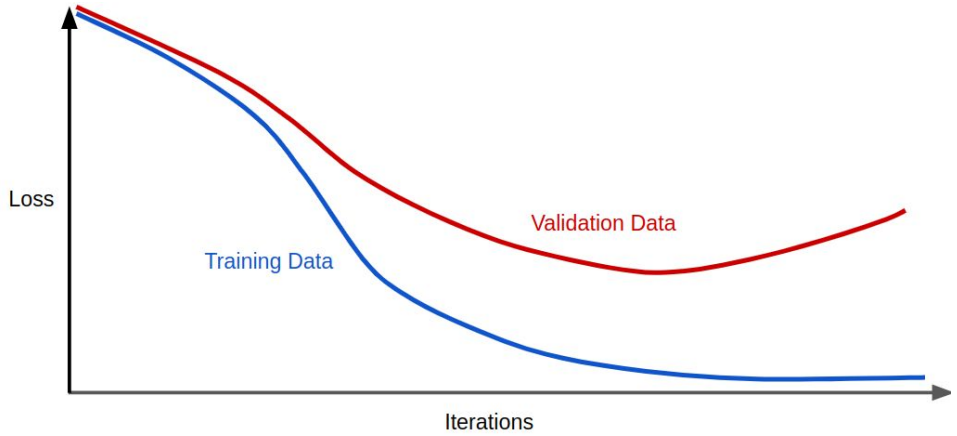
$$y' = b + w1x_1 + w_2x_2 + w_3x_3$$

Não precisamos mudar o modelo para que ele aprenda a calcular w3.

Regularização
Simplicity e Sparsity

Regularização: Simplicity (L2)

- Modelos podem ser **muito complexos.**
- A complexidade muito alta pode contribuir negativamente causando **overfitting.**



Regularização: Simplicity (L2)

Como resolver esse problema?

Punir o modelo de acordo com sua **complexidade.**

$$\text{minimize}(\text{Loss}(\text{Data}|\text{Model})) \rightarrow \text{minimize}\left((\text{Loss}(\text{Data}|\text{Model}) + \text{complexity}(\text{Model})\right)$$

Podemos avaliar a complexidade do modelo de duas formas:

- Complexidade como **função dos pesos**;
- Complexidade relacionada ao número total de **features** diferentes de zero**.**

Regularização: Simplicity (L2)

L2 como função dos pesos:

$$\mathbf{L_2} = ||\mathbf{w}||_2^2 = w_1^2 + w_2^2 + \cdots + w_n^2$$

Pesos **maiores que 1** tem **impacto** muito grande.

Solução: $\text{minimize}\left(\left(\text{Loss}(\text{Data}|\text{Model}\right) + \lambda\,\text{complexity}(\text{Model})\right)$

λ: Taxa de Regularização

→ Tenta jogar os pesos para valores próximos de 0

Regularização: Sparsity (L1)

Modelos de muitas **dimensões** podem demandar bastante RAM.

Isso pode ocorrer de forma natural ou até mesmo na criação de vários **features sintéticos** fazendo-se necessário trabalhar com o que chamamos de **sparse vectors.**

Seria muito conveniente nesses vetores de altas dimensões, conseguirmos **eliminar alguns pesos**, jogando-os para **zero** e assim economizando memória.

L1 x L2

L_1 penaliza |peso| $\rightarrow$ dL_1 penaliza Constante

Subtrai um valor de um peso e quando passa na descontinuidade assume valores nulos.

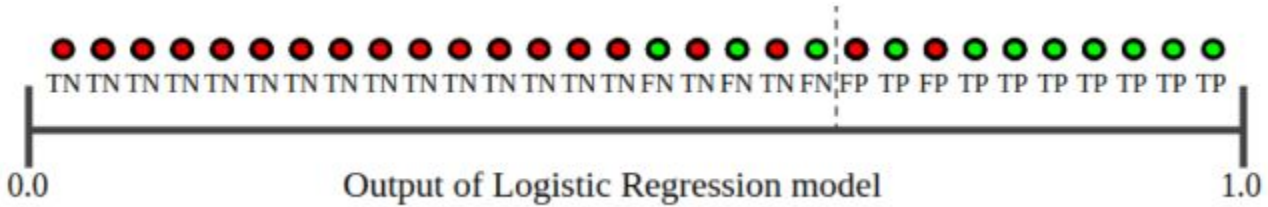
L_2 penaliza peso^2 $\rightarrow$ dL_2 penaliza $2 \cdot$peso

Sempre multiplica o peso por uma constante $|k| \leq 1, k \neq 0$
Tende a zero mas não é zero.

Modelos de Classificação

Thresholding

Os limites de classificação (**Thresholding**)



- Classes positivas e negativas
  Negativo e positivo não está ligado à bom e ruim

Matriz de confusão

Classes positivas e negativas —

- **Verdadeiro positivo** prevê corretamente a classe positiva;
- **Verdadeiro negativo** prevê corretamente a classe negativa;
- **Falso positivo** prevê incorretamente a classe positiva;
- **Falso negativo** prevê incorretamente a classe negativa.

Matriz de confusão

Classes positivas e negativas —

- **Verdadeiro positivo** prevê corretamente a classe positiva **(VP)**;
- **Verdadeiro negativo** prevê corretamente a classe negativa **(VN)**;
- **Falso positivo** prevê incorretamente a classe positiva **(FP)**;
- **Falso negativo** prevê incorretamente a classe negativa **(FN)**.

Accuracy

Métrica de avaliação do modelo:

$$\text{Accuracy} = \frac{\text{Número de previsões corretas}}{\text{Número total de previsões}}$$

Proximidade de estimativa.

Precision x Recall

Precision avalia a margem de acerto em classes **positivas**, relacionando VP e FP**.**

Recall avalia a taxa de acertos, relacionando VP e FN.

$$\text{Precision} = \frac{VP}{VP + FP}$$

$$\text{Recall} = \frac{VP}{VP + FN}$$

Ambos atuam como **métricas de qualidade**, por isso devem ser analisadas em conjunto.

ROC x AUC

Mas como podemos avaliar se o **Threshold** escolhido apresenta os melhores resultados possíveis?

Receiver operator characteristic curva (ROC), avalia a performance em diferentes **limites** no modelo.

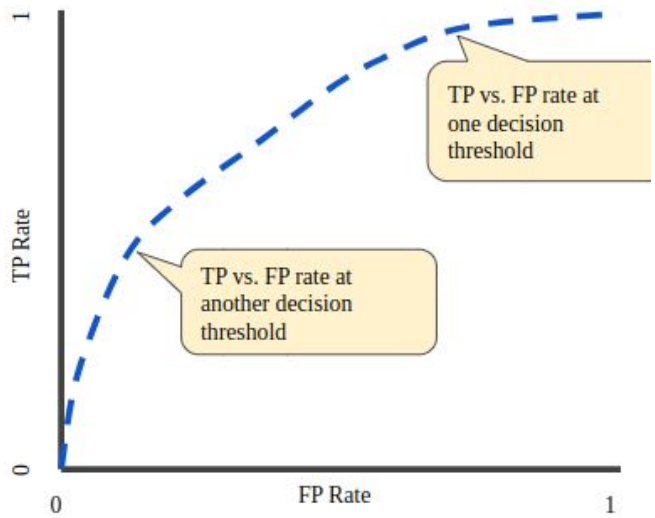
- Taxa de **Verdadeiro Positivo (Recall) — TVP;**
- Taxa de **Falso Positivo — TFP.**

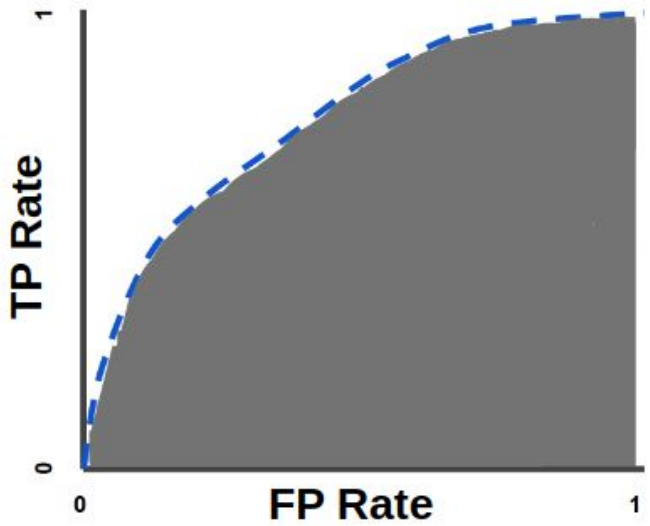
$$TVP = \frac{VP}{VP + FN}$$

$$TFP = \frac{FP}{FP + VN}$$

ROC x AUC

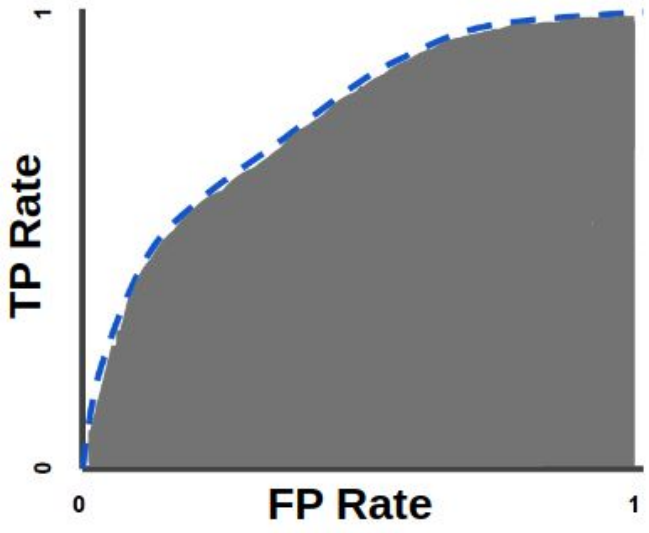
Aplicação do algoritmo **Area Under the Curve (AUC):**





ROC x AUC

Aplicação do algoritmo **Area Under the Curve (AUC):**



Probabilidade do modelo **classificar** o modelo como **positivo** em relação a um **negativo**

Prediction Bias

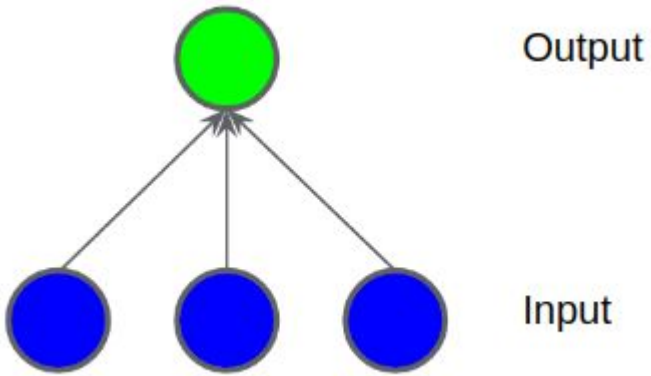
Parâmetro de avaliação da “**eficiência**” do modelo em relação aos classificações feitas e features observados:

$$\text{prediction bias} = \text{average of predticions} - \text{average of labels in data set}$$

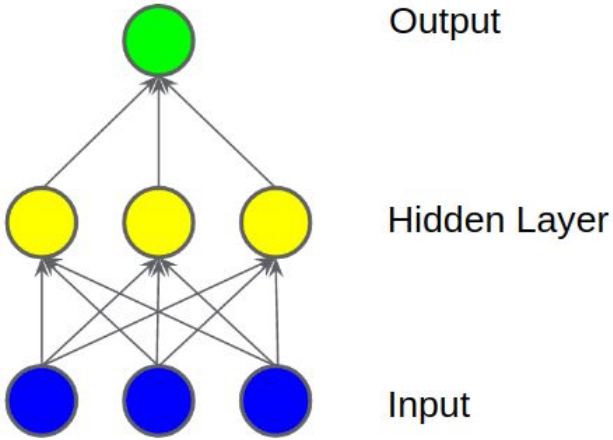
Valores próximos de 0 apresentam ótimos resultados.

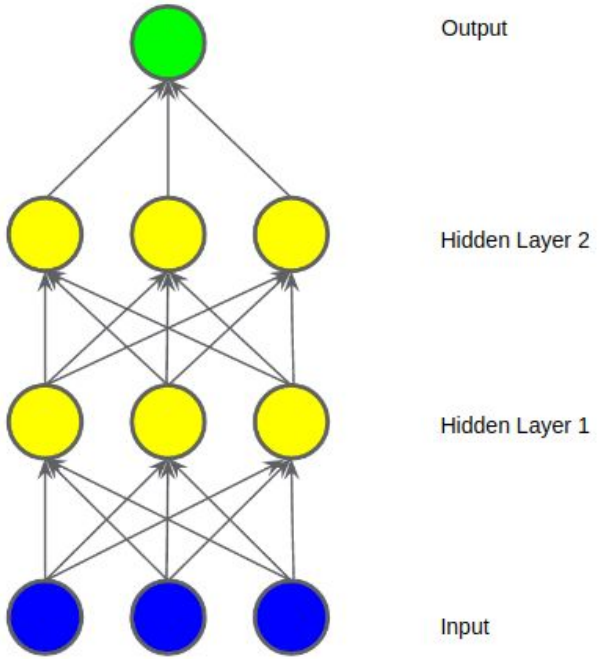
Redes Neurais

Redes Neurais



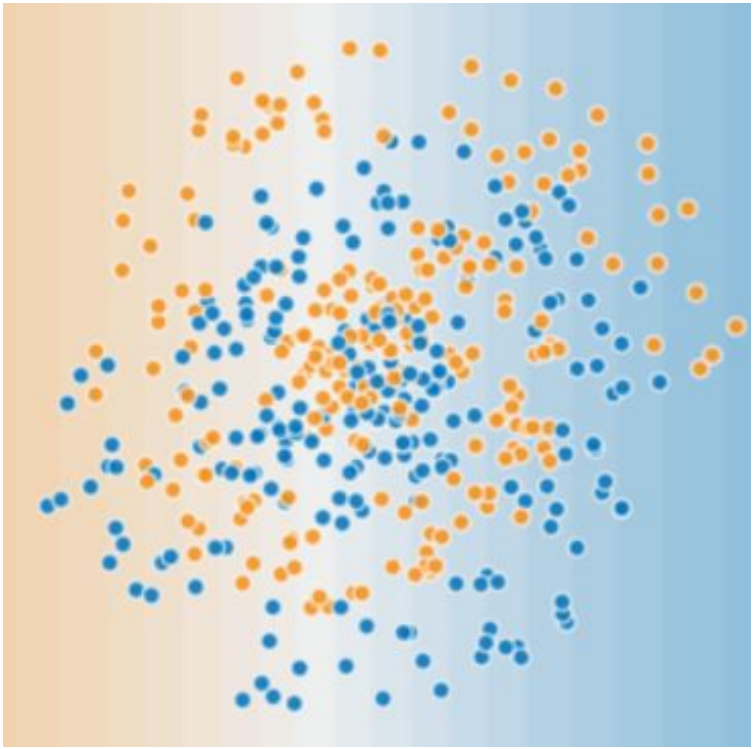
Redes Neurais



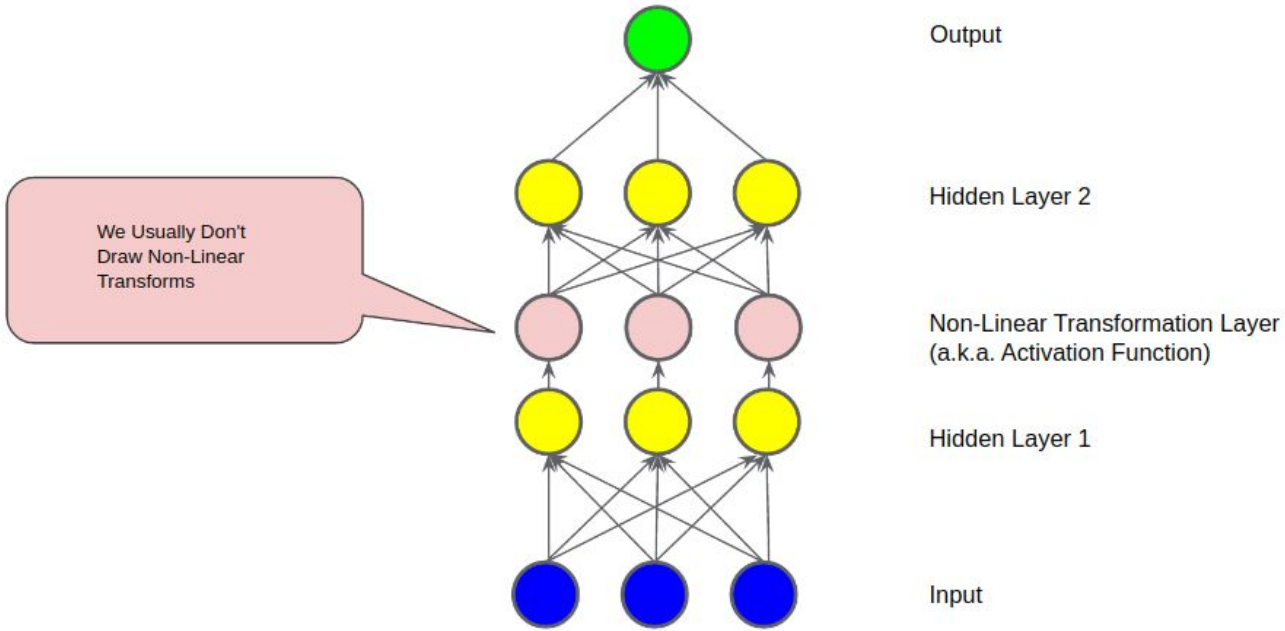


Redes Neurais

Introdução de não linearidade diretamente.



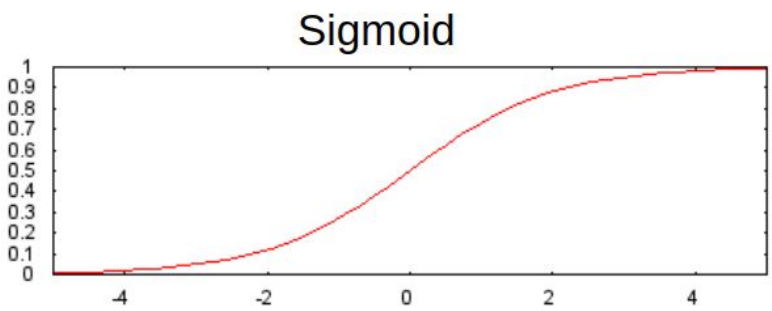
Redes Neurais



Funções de Ativação

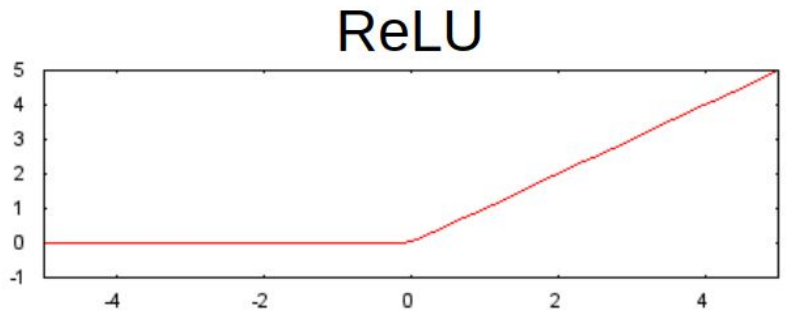
Sigmoid:

$$F(x) = \frac{1}{1 + e^{-x}}$$



Rectified Linear Unit (ReLU):

$$F(x) = max(0, x)$$



Softmax

Softmax tem a função de atribuir **probabilidades** para cada classe em um problema de **muitas classes**.

Class	Probability
apple	0.001
bear	0.04
candy	0.008
dog	0.95
egg	0.001

Softmax

Aplicado na **última camada**, com a restrição de ter o mesmo número de **neurônios** que a **output layer:**

- Função de classificação **completa**;
- Função de classificação **parcial (candidate sampling)**.

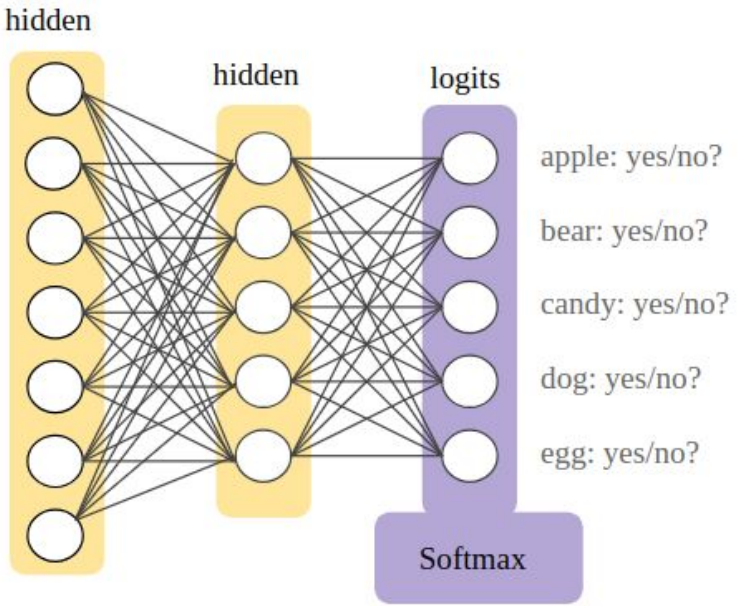


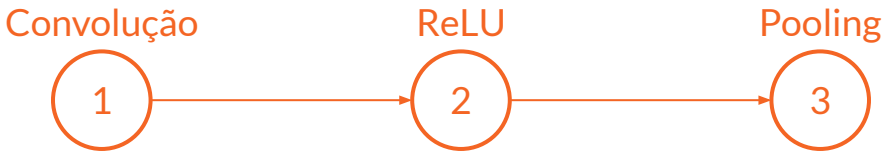
Image Classification

Convolutional Neural Networks

Método de extração de informações de imagens.

Método de aprendizado que em que o modelo consegue aprender **features** de forma autônoma sem necessidade de classificação prévia.

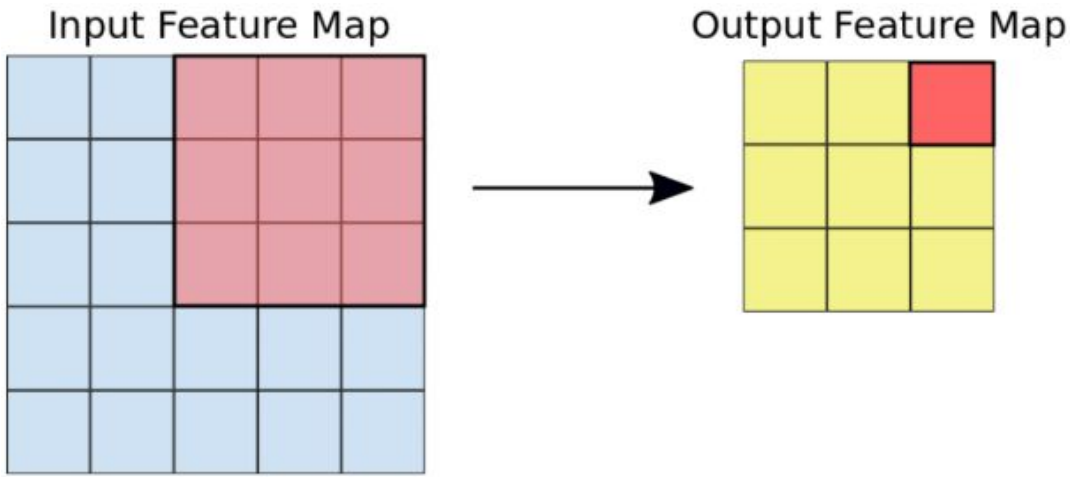
Nossa variável de entrada é chamada de **input feature map** tridimensional que transmite as informações de **comprimento**, **largura** e o último relacionado às **cores**.



1. Convolução

Aplicação de um filtro de tamanho n x n menor que a imagem em janelas da imagem.

A partir dessa aplicação podemos computar mais **features,** resultando em um mapa de **output** chamado **convolved feature.**



1. Convolução

O **convolutional filter** é associado a uma **janela específica** do input. Quanto mais profunda a nossa convolução, mais features conseguimos extrair.

É importante trabalharmos com uma **quantidade** de filtros **otimizada** para nossos problemas para evitar possíveis **gargalos** computacionais no modelo.

Input Feature Map

3	5	2	8	1
9	7	5	4	3
2	0	6	1	6
6	3	7	9	2
1	4	9	5	1

Convolutional Filter

1	0	0
1	1	0
0	0	1

2. ReLU

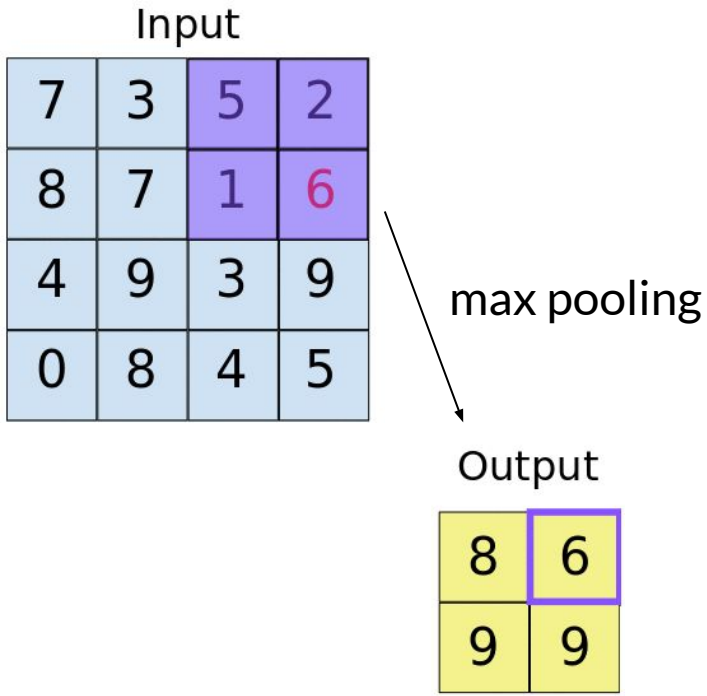
Introdução de não linearidade no sistema através da **Rectified Linear Unit (ReLU)** somada a cada etapa de **convolução.**

$$F(x) = max(0, x) = \begin{cases} 0, & \text{se x} \leq \\ x, & \text{se x} > 0 \end{cases}$$

3. Pooling

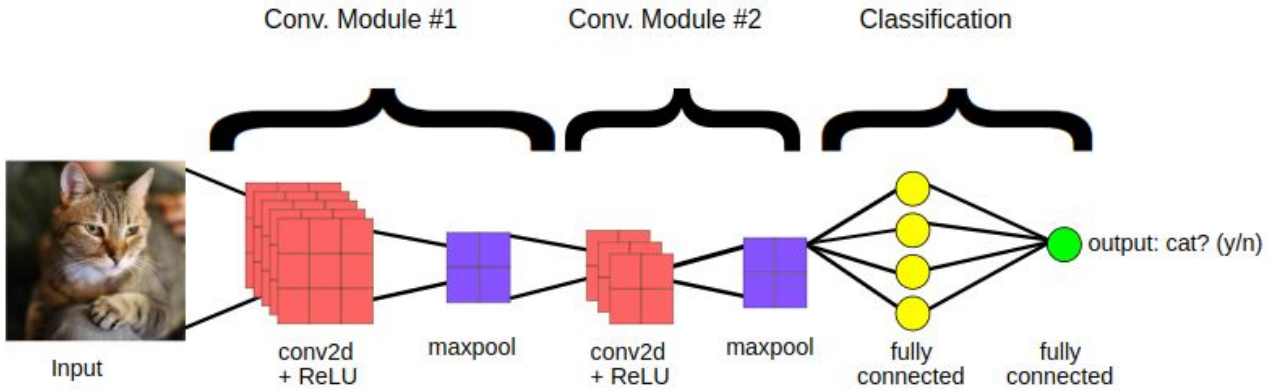
Utilização do algoritmo **max pooling** para processar o sinal depois do **output feature map.**

- Separação de uma nova **janela** onde **guardamos** os **maiores valores** associadas a essa janela.
- Os **parâmetros iniciais** devem ser o **tamanho** do filtro e a **distância** de cada janela.



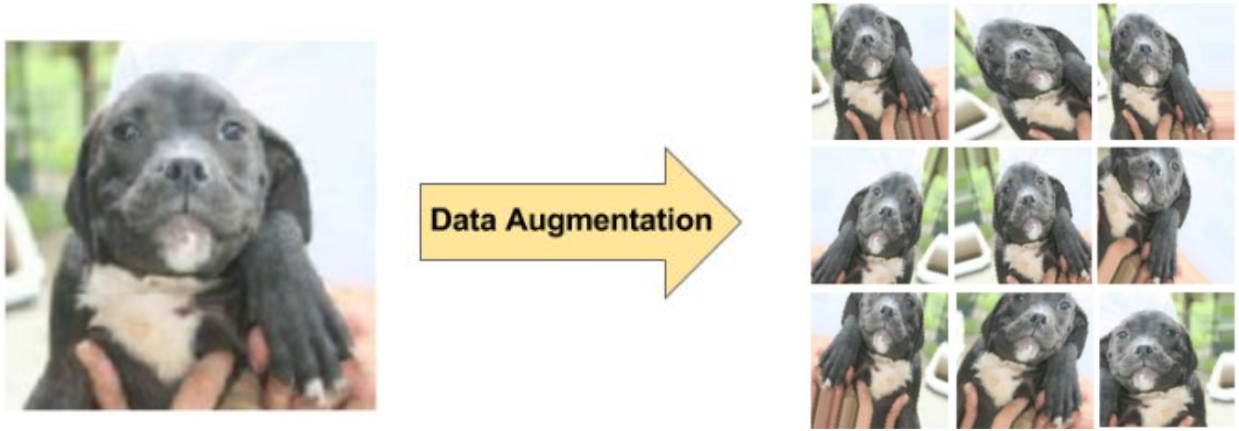
Fully Connected Layers

Normalmente a **última camada** do modelo, responsável por **atribuir** uma **probabilidade** de classificação com base nos **features extraídos** nas camadas convolucionais por meio de uma função **softmax**.



Augmentation

Existem alguns métodos internos que nos ajudam a evitar o **overfitting.**



Aumentam a capacidade de aprendizado do modelo utilizando o mesmo banco de dados.

Modelos Pré treinados

Como modelos de treinamento podem demandar bastante tempo finalizar, podemos usar **modelos pré-treinados** e **adaptá-los** segundo as nossas **necessidades**.

Uma forma de usar modelos pré-treinados é a partir da **extração de features**. Essa extração **retira** representações **da última cama de convolução** de um modelo pré-treinado e depois **usa essas representações como inputs em novos modelos.**