Generative AI em Java

Generative AI é um ramo avançado da inteligência artificial que cria novos dados sintéticos, como texto, imagens, áudio ou vídeo, com base em padrões aprendidos de grandes volumes de dados.

As principais abordagens técnicas incluem:

- **Transformers:** Modelos como GPT que utilizam mecanismos de atenção para processar sequências de dados, permitindo a geração de conteúdo contextualizado e fluido.

- **Redes Generativas Adversariais (GANs):** Compostas por dois componentes, um gerador que cria dados e um discriminador que avalia a autenticidade desses dados. O gerador melhora continuamente, buscando enganar o discriminador e produzindo amostras cada vez mais realistas.

- **Autoencoders Variacionais (VAEs):** Modelos que aprendem a representar dados em um espaço latente comprimido e geram novas amostras variando pontos nesse espaço, comuns na geração de imagens e áudio.

- Processo de Treinamento: Envolve otimização através de backpropagation e gradiente estocástico, com funções de perda específicas para avaliar a qualidade das saídas geradas em relação aos dados reais.

- **Amostragem (Sampling):** Métodos como Top-K Sampling e Nucleus Sampling são usados para ajustar a diversidade e qualidade das saídas, controlando o equilíbrio entre criatividade e coerência.

Generative AI aprende distribuições probabilísticas complexas, permitindo criar dados novos e originais, com aplicações que vão desde geração de linguagem natural e imagens fotorrealistas até síntese de áudio e vídeo interativo.

Para o exemplo, vamos usar a biblioteca DeepLearning4J (DL4J), uma biblioteca poderosa de aprendizado profundo para Java.

A ideia é criarmos especificamente uma Rede Neural Generativa Simples (Multilayer Perceptron), que pode ser usada para gerar saídas com base em entradas aprendidas.

Dependências:

Você precisará incluir as seguintes dependências no seu pom.xml se estiver usando o Maven:

```
<dependencies>
    <dependency>
        <groupId>org.deeplearning4j</groupId>
        <artifactId>deeplearning4j-core</artifactId>
        <version>1.0.0-M1.1</version>
    </dependency>
    <dependency>
        <groupId>org.nd4j</groupId>
        <artifactId>nd4j-native-platform</artifactId>
        <version>1.0.0-M1.1</version>
    </dependency>
</dependencies>
```

Exemplo de Código:

Aqui está um exemplo simples de como criar uma rede neural gerativa com Java, usando o DeepLearning4J:

```
Import org.deeplearning4j.nn.api.OptimizationAlgorithm;
Import org.deeplearning4j.nn.conf.MultiLayerConfiguration;
Import org.deeplearning4j.nn.conf.NeuralNetConfiguration;
Import org.deeplearning4j.nn.conf.layers.OutputLayer;
Import org.deeplearning4j.nn.conf.layers.DenseLayer;
Import org.deeplearning4j.nn.multilayer.MultiLayerNetwork;
Import org.deeplearning4j.optimize.listeners.ScoreIterationListener;
Import org.nd4j.linalg.activations.Activation;
Import org.nd4j.linalg.dataset.DataSet;
Import org.nd4j.linalg.dataset.api.iterator.impl.ListDataSetIterator;
Import org.nd4j.linalg.factory.Nd4j;
Import org.nd4j.linalg.learning.config.Sgd;
Import org.nd4j.linalg.lossfunctions.LossFunctions;
Import java.util.Arrays;
Import java.util.List;

Public class SimpleGenerativeAI {

    Public static void main(String[] args) {

        // Configurando a rede neural gerativa simples
        Int inputSize = 2; // Tamanho da entrada
        Int outputSize = 1; // Tamanho da saída

        MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
                .seed(12345) // Para reproduzir os resultados
                .optimizationAlgo(OptimizationAlgorithm.STOCHASTIC_GRADIENT_DESCENT)
                .updater(new Sgd(0.1)) // Usando SGD como otimizador
                .list()
                .layer(0, new DenseLayer.Builder().nIn(inputSize).nOut(3)
                        .activation(Activation.RELU)
                        .build())
                .layer(1, new OutputLayer.Builder(LossFunctions.LossFunction.MSE)
                        .activation(Activation.IDENTITY)
                        .nIn(3).nOut(outputSize).build())
                .build();
```

```
        // Criando a rede neural
        MultiLayerNetwork network = new MultiLayerNetwork(conf);
        Network.init();
        Network.setListeners(new ScoreIterationListener(10));

        // Conjunto de dados de exemplo (entradas e saídas)
        Double[][] input = new double[][]{
                {0.0, 0.0}, {0.0, 1.0},
                {1.0, 0.0}, {1.0, 1.0}
        };

        Double[][] output = new double[][]{
                {0.0}, {1.0},
                {1.0}, {0.0}
        };

        List<DataSet> data = Arrays.asList(new DataSet(Nd4j.create(input), Nd4j.create(output)));
        ListDataSetIterator<DataSet> dataSetIterator = new ListDataSetIterator<>(data, 1);

        // Treinando a rede neural
        For (int i = 0; i < 1000; i++) {
           dataSetIterator.reset();
           network.fit(dataSetIterator);
        }

        // Testando a rede neural gerativa
        Double[] testInput = {0.5, 0.5}; // Entrada de teste
        Double[] predicted = network.output(Nd4j.create(testInput)).toDoubleVector();

        System.out.println("Entrada: " + Arrays.toString(testInput));
        System.out.println("Saída Gerada: " + Arrays.toString(predicted));
    }
}
```

Explicação:

- **DenseLayer**: Camada densa que faz parte da rede neural. Aqui, temos uma camada com 3 neurônios.
- **OutputLayer**: Camada de saída que usa uma função de perda de erro quadrático médio (MSE) e uma ativação linear (IDENTITY).
- **SGD (Stochastic Gradient Descent):** Método de otimização escolhido para ajustar os pesos da rede neural.
- **Treinamento**: A rede neural é treinada para aprender a relação entre os valores de entrada e saída.

Execução:

Quando você executar este código, ele treinará uma rede neural simples que tenta aprender uma função XOR (valores binários de entrada e saída). Você pode modificar a rede ou os dados para adaptá-la ao seu problema de geração.

Esse é um exemplo básico, mas serve como ponto de partida para explorar redes generativas mais complexas, como GANs (Generative Adversarial Networks).

Exemplo de Generative AI Didática

Processamento de Matrizes com EJML em Java

Este documento técnico descreve o procedimento para criar, transpor e multiplicar matrizes utilizando a biblioteca EJML (Efficient Java Matrix Library) em Java. A EJML é uma biblioteca para operações de álgebra linear em Java, oferecendo suporte para manipulação e cálculo com matrizes.

Objetivo

O objetivo deste documento é demonstrar como realizar as seguintes operações com matrizes usando EJML:

a) *Criação de matrizes a partir de dados brutos.*
b) *Transposição de uma matriz.*
c) *Multiplicação de matrizes.*

Pré-requisitos

a) Conhecimento básico de álgebra linear.
b) Ambiente de desenvolvimento Java configurado.
c) Biblioteca EJML incluída no projeto.

Procedimento

1. Configuração do Projeto

Certifique-se de que a biblioteca EJML está incluída no seu projeto Java. Você pode adicionar a dependência EJML ao seu projeto Maven ou baixar o JAR diretamente.

Para Maven, adicione o seguinte ao seu pom.xml:

```
<dependency>
    <groupId>org.ejml</groupId>
    <artifactId>ejml-simple</artifactId>
    <version>0.42</version>
</dependency>
```

2. Criação de Matrizes

A criação de matrizes envolve definir os dados e instanciar objetos SimpleMatrix da EJML.

```
import org.ejml.simple.SimpleMatrix;

public class MatrixOperations {
    public static void main(String[] args) {
        // Dados para a matriz de entrada
        double[][] input = {
            {1.0, 2.0, 3.0},
            {2.0, 3.0, 4.0},
            {3.0, 4.0, 5.0},
            {4.0, 5.0, 6.0}
        };

        // Criação da matriz com EJML
        SimpleMatrix inputMatrix = new SimpleMatrix(input);
    }
}
```

3. Transposição da Matriz

A transposição de uma matriz é feita usando o método transpose() da classe SimpleMatrix.

```
SimpleMatrix transposedMatrix = inputMatrix.transpose();
```

4. Multiplicação de Matrizes

Para multiplicar a matriz transposta pela matriz original, utilize o método mult().

```
SimpleMatrix resultMatrix = transposedMatrix.mult(inputMatrix);
```

5. Exibição dos Resultados

Para exibir as matrizes e o resultado de forma legível, você pode implementar uma função auxiliar para formatar a saída.

```
private static void printMatrix(SimpleMatrix matrix) {
    int rows = matrix.numRows();
    int cols = matrix.numCols();
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            System.out.printf("%10.4f ", matrix.get(i, j));
        }
        System.out.println();
    }
}
```

Exemplo Completo

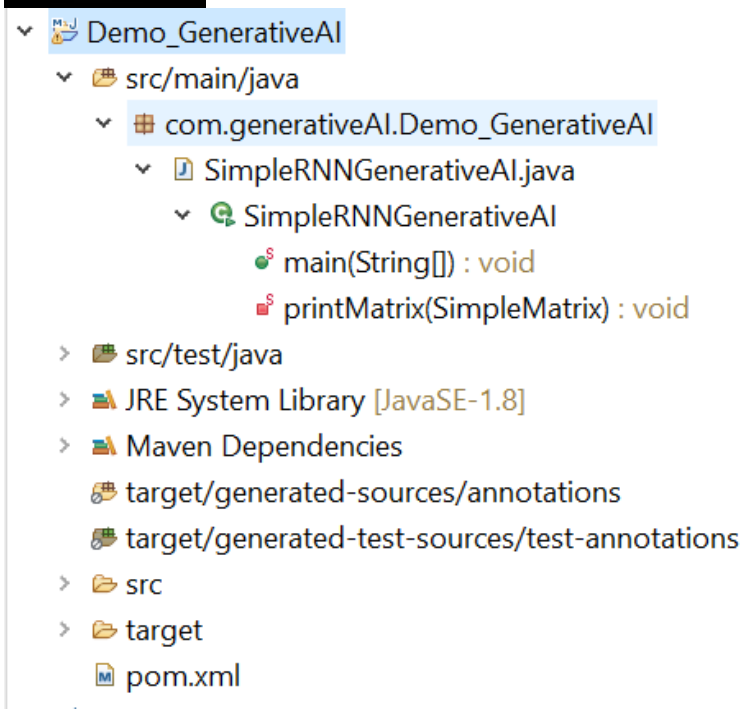
```
package com.generativeAI.Demo_GenerativeAI;

import org.ejml.simple.SimpleMatrix;

public class SimpleRNNGenerativeAI {
    public static void main(String[] args) {
        // Definindo as matrizes de entrada e saída
        double[][] input = {
            {1.0, 2.0, 3.0},
            {2.0, 3.0, 4.0},
            {3.0, 4.0, 5.0},
            {4.0, 5.0, 6.0}
        };

        SimpleMatrix inputMatrix = new SimpleMatrix(input);

        // Exemplo de operação: transposição e multiplicação
        SimpleMatrix transposedInput = inputMatrix.transpose();
        SimpleMatrix result = transposedInput.mult(inputMatrix);

        // Exibindo resultados
        System.out.println("Entrada:");
        printMatrix(inputMatrix);
        System.out.println("Resultado da multiplicação e transposição:");
        printMatrix(result);
    }

    private static void printMatrix(SimpleMatrix matrix) {
        int rows = matrix.numRows();
        int cols = matrix.numCols();
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                System.out.printf("%10.4f ", matrix.get(i, j));
            }
            System.out.println();
        }
    }
}
```

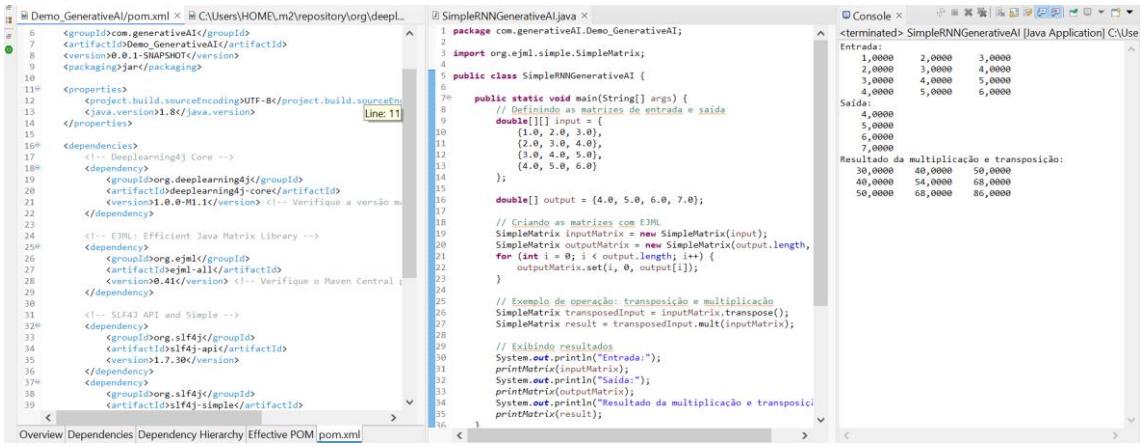
Conclusão

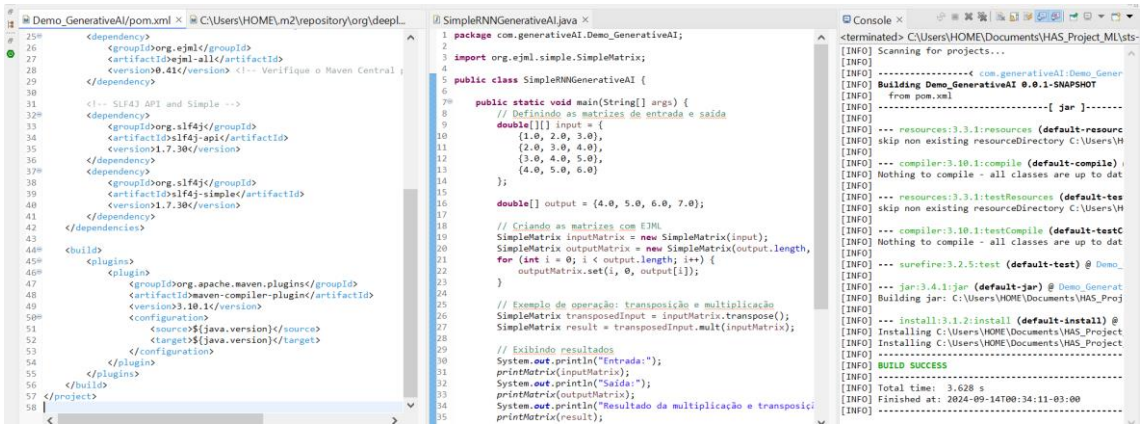
Este documento detalha o processo de criação, transposição e multiplicação de matrizes usando a biblioteca EJML em Java. A EJML fornece uma API simples e eficiente para operar com matrizes, facilitando a realização de cálculos de álgebra linear.

Classe Completa SimpleRNNGenerativeAI

```java
package com.generativeAI.Demo_GenerativeAI;

import org.ejml.simple.SimpleMatrix;

public class SimpleRNNGenerativeAI {

    public static void main(String[] args) {
        // Definindo as matrizes de entrada e saída
        double[][] input = {
            {1.0, 2.0, 3.0},
            {2.0, 3.0, 4.0},
            {3.0, 4.0, 5.0},
            {4.0, 5.0, 6.0}
        };

        double[] output = {4.0, 5.0, 6.0, 7.0};

        // Criando as matrizes com EJML
        SimpleMatrix inputMatrix = new SimpleMatrix(input);
        SimpleMatrix outputMatrix = new SimpleMatrix(output.length, 1);
        for (int i = 0; i < output.length; i++) {
            outputMatrix.set(i, 0, output[i]);
        }

        // Exemplo de operação: transposição e multiplicação
        SimpleMatrix transposedInput = inputMatrix.transpose();
        SimpleMatrix result = transposedInput.mult(inputMatrix);

        // Exibindo resultados
        System.out.println("Entrada:");
        printMatrix(inputMatrix);
        System.out.println("Saída:");
        printMatrix(outputMatrix);
        System.out.println("Resultado da multiplicação e transposição:");
        printMatrix(result);
    }

    // Função para imprimir uma matriz formatada
    private static void printMatrix(SimpleMatrix matrix) {
        int rows = matrix.numRows();
        int cols = matrix.numCols();
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                System.out.printf("%10.4f ", matrix.get(i, j));
            }
            System.out.println();
        }
    }
}
```


Demo_GenerativeAI
src/main/java
com.generativeAI.Demo_GenerativeAI
SimpleRNNGenerativeAI.java
SimpleRNNGenerativeAI
main(String[]) : void
printMatrix(SimpleMatrix) : void
src/test/java
JRE System Library [JavaSE-1.8]
Maven Dependencies
target/generated-sources/annotations
target/generated-test-sources/test-annotations
src
target
pom.xml


Demo_GenerativeAI/pom.xml
C:\Users\HOME\.m2\repository\org\deepl...
SimpleRNNGenerativeAI.java
Console
<terminated> SimpleRNNGenerativeAI [Java Application] C:\Use
Entrada:
1,0000 2,0000 3,0000
2,0000 3,0000 4,0000
3,0000 4,0000 5,0000
4,0000 5,0000 6,0000
Saída:
4,0000
5,0000
6,0000
7,0000
Resultado da multiplicação e transposição:
30,0000 40,0000 50,0000
40,0000 54,0000 68,0000
50,0000 68,0000 86,0000
Overview
Dependencies
Dependency Hierarchy
Effective POM
pom.xml


Demo_GenerativeAI/pom.xml
C:\Users\HOME\.m2\repository\org\deepl...
SimpleRNNGenerativeAI.java
Console
<terminated> C:\Users\HOME\Documents\HAS_Project_ML\sts-
[INFO] Scanning for projects...
[INFO] Building Demo_GenerativeAI 0.0.1-SNAPSHOT
[INFO] BUILD SUCCESS
[INFO] Total time: 3.628 s
[INFO] Finished at: 2024-09-14T00:34:11-03:00