# Letters MLP

`Projeto da disciplina de Inteligência Artificial (EACH-USP)`

O projeto consiste na implementação e treinamento de uma rede neural artificial do tipo Multilayer Perceptron (MLP) em Java, sem o uso de bibliotecas especializadas. O objetivo é criar uma estrutura base de MLP capaz de receber imagens das 26 letras do alfabeto e identificar qual letra está sendo enviada.

## Estrutura

- `src`: arquivos .java da estrutura da rede neural.
- `run`: arquivos .java que processam o treinamento da rede (com e sem Cross Validation e Parada Antecipada).
- `files`: arquivos que serão usados no treinamento da rede neural.
- `data`: arquivos separados e organizados para o treinamento.
- `plot`: arquivos .py que fazem a plotagem dos gráficos da Matriz de Confusão e do MSE.
- `bin`: arquivos .class.

## Classes

### NeuralNetwork

A classe `NeuralNetwork` representa toda a estrutura da rede neural, composta por `n` camadas e `m` neurônios dentro de cada camada. Ela fornece métodos para realizar o treinamento e processamento da rede. - **Atributos**: - `layers`: Array de camadas na rede neural. - `layerInfo`: Array utilizado pra inicializar a rede. Cada elemento `i` representa uma camada e cada `valor` de [i] representa a quantidade de neurônios que a camada terá. - `inputs`: Array de inputs que a rede recebe para rodar o feedforward. - `finalOutputs`: Array de outputs finais após rodar o feedforward. - `expectedOutputs`: Array de outputs esperados para rodar o backpropagation. - `MSE`: Valor do erro quadrático médio.

- **Métodos**:
  - `NeuralNetwork(int[] layerSizes)`: Construtor para inicializar a rede neural com as camadas e números de neurônios especificados.
  - `double[] feedForward(double[] inputs)`: Realiza o `feedforward` na rede neural com base nas entradas fornecidas.
  - `void backpropagation(double[] expectedOutputs, double learningRate)`: Realiza o `backpropagation` na rede neural com base nas saídas esperadas e no larning rate.
  - `void calculateMSE(double[] expectedOutputs)`: Calcula o Erro Quadrático Médio com base em finalOutputs e expectedOutputs.
  - `double[] getOutputs(double[] inputs)`: Devolve os outputs da camada de saída com base nas entradas fornecidas.

### Layer

A classe `Layer` representa uma camada da rede neural. - **Atributos**: - `layerIndex`: Índice da camada. - `previousLayer`: Instância Layer da camada anterior. - `nextLayer`: Instância Layer da próxima camada. - `neurons`: Array de neurônios na camada. - `outputs`: Array de outputs de cada neurônio.

- **Métodos**:
  - `Layer(int layerIndex, int numNeurons, int numInputsPerNeuron)`: Construtor para inicializar a camada e os seus neurônios.
  - `double[] calculateOutputs(double[] inputs)`: Realiza os cálculos do `feedforward` considerando se é uma camada de entrada ou oculta.
  - `void backpropagate(double[] expectedOutputs, double learningRate)`: Realiza os cálculos do `backpropagation` considerando se é uma camada de saída ou oculta.
  - `void updateWeightsAndBiases()`: Atualiza os pesos e bias de todos os neurônios da camda com base nos deltas calculdos no backpropagate.

### Neuron

A classe `Neuron` representa um neurônio da uma rede neural. - **Atributos**: - `neuronIndex`: Índice do neurônio na camada. - `layerIndex`: Índice da camada a qual o neurônio pertence. - `inWeights`: Array de pesos de entrada do neurônio. - `bias`: Bias do neurônio. - `inputs`: Array de inputs do neurônio. - `sum`: Somatória dos inputs multiplicados pelos respectivos pesos. - `output`: Saída do neurônio após o método `calculateOutput`. - `errorInfo`: Informação de erro de cada neurônio com base no expectedOutput. - `delta`: Array de deltas relacionado cada peso do neurônio. - `biasDelta`: Delta relacionado ao bias.

- **Métodos**:
  - `Neuron(int layerIndex, int neuronIndex, int numInputs)`: Construtor para inicializar os neurônios com pesos aleatórios.
  - `double calculateOutput(double[] inputs)`: Calcula a saída do neurônio com base nas entradas, pesos e bias.
  - `void updateWeightsAndBias()`: Atualiza os pesos e o bias com base no delta.
  - `double outputGradient(double expectedOutput)`: Calcula o valor esperado - output.
  - `double sigmoidDerivative()`: Calcula da derivada da sigmóide com base no sum.
  - `double sigmoid(double x)`: Função de ativação sigmoidal.