

Experimentação Redes Neurais

Pedro Henrique Honorio Saito

122149392


1. Separação do Dataset em Treinamento e Teste

Para realizar a separação do *dataset* em teste e treinamento como solicitado, começaremos carregando o *dataset* IRIS e estratificando-o da seguinte forma:

```
from sklearn.datasets import load_iris

iris = load_iris()
X = iris.data
y = iris.target

X_train_full, X_test, y_train_full, y_test = train_test_split(
    X, y,
    test_size=0.2,
    stratify=y,
    random_state=42,
)
```

 Python

Com isso, garantimos que os dados sejam agrupados de modo a refletir a porcentagem de cada classe proveniente do *dataset* original. Aqui estão os resultados:


- Obtivemos 120 instâncias (80%) para treino e 30 instâncias (20%) para teste.
- Cada subconjunto possui a mesma quantidade de amostras de cada classe (32 por classe no treino e 8 por classe no teste).

2. Validação Cruzada com *K-Fold* ($K = 5$)

Nos 80% dos dados de treino, aplicamos o *Stratified K-Fold* com $K = 5$ para avaliar as diferentes arquiteturas da rede neural:

```
from sklearn.model_selection import StratifiedKFold

skf = StratifiedKFold(n_split=5, shuffle=True, random_state=42)
```

 Python

Para cada partição gerada, dividimos em:

- **Conjunto de Treino (5 folds - 1).**
- **Conjunto de Validação (1 fold).**

Portanto, mantemos sempre o balanceamento estratificado em cada fold.

3. Pré-processamento (Todas as Variáveis de Entrada)

O *dataset IRIS* é homogêneo o suficiente para o uso direto do `MLPClassifier` e, desse modo, não aplicamos técnicas de padronização e normalização de forma explícita. Assim, utilizamos todas as variáveis de entrada (*sepal length*, *sepal width*, *petal length* e *petal width*) sem modificações adicionais.

4. Configuração Inicial do `MLPClassifier`


Para cada configuração, fixamos:

- **Solver:** `sgd`.
- **Parâmetros Default do `MLPClassifier`.**
- **Arquitetura inicial:** `hidden_layer_sizes=(100,)`, isto é, uma camada oculta de 100 neurônios.
- **Funções de Ativação Testadas:** `logistic`, `tanh` e `relu`.

Aqui está o código referente a essas adições:


```
from sklearn.neural_network import MLPClassifier

activations = ['logistic', 'tanh', 'relu']
hidden_layers_initial = (100,)
solver = 'sgd'
```

 Python

Além disso, para cada função de ativação a , e a arquitetura acima, executamos o seguinte pseudocódigo na forma de *loop*:

```
para cada fold em StratifiedKFold(K=5):
    - Treinar MLPClassifier(
        activation=a, solver='sgd', hidden_layer_sizes=(100,), random_state=42
    ) no conjunto de treino do fold
    - Computar acurácia de:
        - Conjunto de Treino do fold (score_de_treinamento)
        - Conjunto de Validação do fold (score_de_validação)
    - Salvar curva de perda (clf.loss_curve_)
```

 Python

5. Resultados do Experimento Inicial

5.1. Scores de Treinamento e Validação por Fold

Aqui estão alguns dos resultados obtidos:

Ativação	Fold	Acurácia Treino	Acurácia Validação
logistic	1	0,677	0,667
	2	0,677	0,667
	3	0,688	0,625
	4	0,698	0,667
	5	0,677	0,750
	Média	0,683	0,675

Tabela 1: Métricas para função de ativação logística.

Ativação	Fold	Acurácia Treino	Acurácia Validação
tanh	1	0,969	0,958
	2	0,969	0,958
	3	0,969	1,000
	4	0,969	1,000
	5	0,979	0,917
	Média	0,971	0,967

Tabela 2: Métricas para função de ativação tangente hiperbólica.

Ativação	Fold	Acurácia Treino	Acurácia Validação
relu	1	0,938	0,917
	2	0,948	0,875
	3	0,906	1,000
	4	0,927	0,958
	5	0,927	0,917
	Média	0,929	0,933

Tabela 3: Métricas para função de ativação relu.

Como podemos constatar:

- A função de ativação tanh obteve a maior acurácia média de validação (0,967).
- relu alcançou uma média de validação de 0,933 e logistic ficou em 0,675.
- O espaço entre treino e validação ficou pequeno para tanh e relu, indicando um bom ajuste.

5.2. Comportamento da Função de Perda

No mesmo gráfico, plotamos todas as curvas de perda para cada fold e cada ativação:

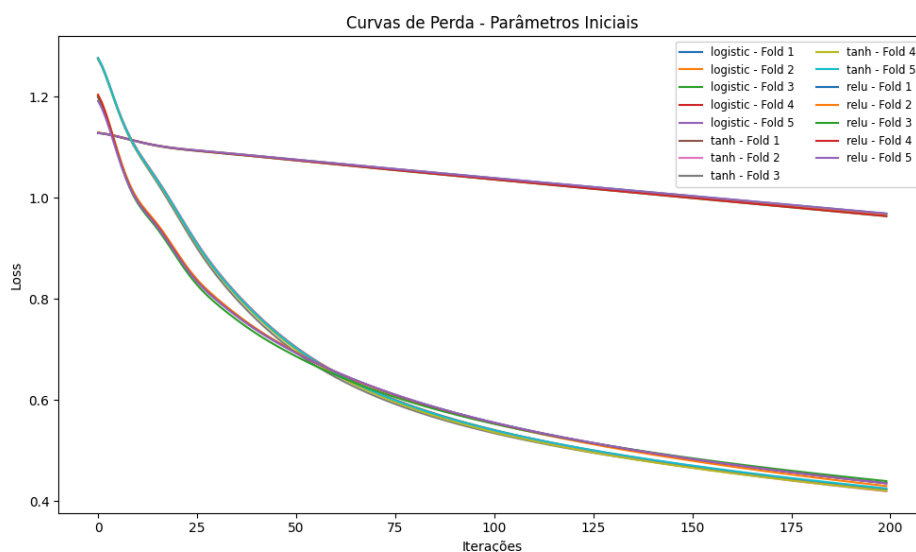


Figura 1: Curvas de perda com parâmetros iniciais.

Aqui estão alguns dos parâmetros iniciais das curvas de perda:

- **relu**: Perda despenca nas iterações iniciais, mas exibe flutuações moderadas conforme se aproxima do mínimo.
- **tanh**: Trajetória de perda suave, converge rapidamente e atinge níveis mais baixos sem oscilações significativas.
- **logistic**:

6. Análise dos Resultados Iniciais

1. Comparação das Ativações:

- **tanh** mostrou-se a mais estável e a que melhor generalizou (acurácia de validação 0,967).
- **relu** obteve boa performance, mas média de validação (0,933) ficou abaixo de **tanh**.
- **logistic** apresentou resultado bem inferior (0,675), indicando que, sem ajustes de parâmetros adicionais, sigmóides simples não foram adequados para esse cenário.

2. Convergência e Loss:

- **tanh** convergiu em poucos ciclos do SGD sem grandes oscilações.
- **relu** convergiu mais rápido nas primeiras iterações, mas oscilou ao final.
- **logistic** mostrou instabilidade, sugerindo necessidade de maior número de iterações ou *learning rate* menor se fosse mantido.

1. Overfitting e Gap entre Treino/Validação

- **tanh** e **relu** exibiram gap reduzido entre treino e validação, indicando um bom ajuste sem overfitting severo.
- Por outro lado, **logistic** teve gap menor, mas ambas as acurácias (treino e validação) ficaram baixas, indicando underfitting.

7. Experimento com Parâmetros Modificados

Ainda usando os 80% dos dados e $K\text{-Fold} = 5$, repetimos o experimento alterando:


- **hidden_layer_sizes=(50, 50)**: Possui objetivo de adicionar profundidade para capturar padrões mais complexos.
- **learning_rate_init=0.01**: Permite acelerar a convergência do SGD, mesmo com o risco de maior oscilação.

Aqui está a implementação dessa parte:

```
hidden_layers_altered = (50, 50)
learning_rate_init_altered = 0.01
results_altered = {}

for act in activations:
    train_scores, val_scores, loss_curves = [], [], []
    for train_idx, val_idx in skf.split(X_train_full, y_train_full):
        X_train, X_val = X_train_full[train_idx], X_train_full[val_idx]
        y_train, y_val = y_train_full[train_idx], y_train_full[val_idx]

        clf = MLPClassifier(
```

 Python

```

        activation=act,
        solver=solver,
        hidden_layer_sizes=hidden_layers_altered,
        learning_rate_init=learning_rate_init_altered,
        random_state=42
    )
    clf.fit(X_train, y_train)

    train_scores.append(accuracy_score(y_train, clf.predict(X_train)))
    val_scores.append(accuracy_score(y_val, clf.predict(X_val)))
    loss_curves.append(clf.loss_curve_)

results_altered[act] = {
    'train_scores': train_scores,
    'val_scores': val_scores,
    'loss_curves': loss_curves
}

```

7.1. Scores de Treinamento e Validação por Fold (Parâmetros Modificados)

Ativação	Fold	Acurácia Treino	Acurácia Validação
logistic	1	0,333	0,333
	2	0,333	0,333
	3	0,333	0,333
	4	0,333	0,333
	5	0,333	0,333
	Média	0,333	0,333

Tabela 4: Métricas para função de ativação logística modificada.

Ativação	Fold	Acurácia Treino	Acurácia Validação
tanh	1	0,979	0,958
	2	0,979	0,958
	3	0,990	1,000
	4	0,990	0,958
	5	0,990	0,917
	Média	0,985	0,967

Tabela 5: Métricas para função de ativação tangente hiperbólica modificada.

Ativação	Fold	Acurácia Treino	Acurácia Validação
relu	1	0,979	0,958
	2	0,979	1,000
	3	0,979	1,000
	4	0,990	1,000
	5	0,990	0,917
	Média	0,983	0,975

Tabela 6: Métricas para função de ativação relu modificada.

Aqui podemos perceber que:

- relu atingiu a maior média de validação (0,975), superando tanh (0,967).
- tanh manteve a média igual à do experimento inicial (0,967).
- logistic obteve desempenho bem baixo, indicando falha durante a aprendizagem com esses parâmetros.

7.2. Curvas de Perda (Parâmetros Alterados)

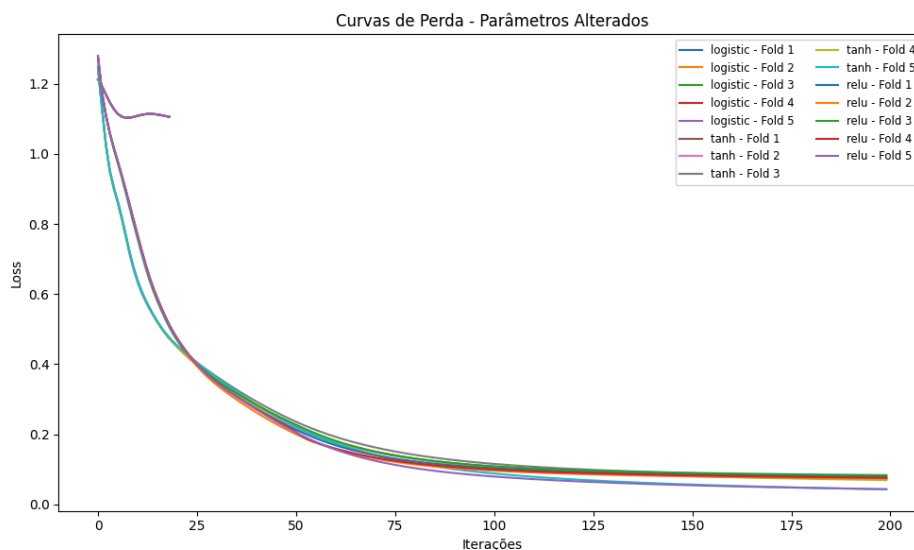


Figura 2: Curvas de perda com parâmetros alterados.

Os parâmetros alterados usados na geração das curvas de perda foram os seguintes:

- Com relu, a perda despencou rapidamente em cada fold e atingiu valores baixos em poucas iterações, justificando a melhora na validação.
- tanh manteve a trajetória suave, porém a *learning rate* maior acelerou ligeiramente a convergência.
- logistic manteve patamar de *loss* elevado, não aprendendo nada (acurácia de 0,333 fixa).


8. Escolha do Melhor Modelo e Treinamento Final

Comparando a média de validação entre os experimentos concluímos que:

- Melhor média **inicial**: tanh com 0,967.
- Melhor média **alterado**: relu com 0,975.


Desse modo, o melhor **modelo global** foi obtido com:

```
MLPClassifier(
    activation='relu',
    solver='sgd',
    hidden_layer_sizes=(50, 50),
    learning_rate_init=0.01,
    random_state=42
)
```

 Python

Treinamos esse modelo usando todos os 120 exemplos de treino:

```
clf_best = MLPClassifier(
    activation='relu',
    solver='sgd',
    hidden_layer_sizes=(50, 50),
    learning_rate_init=0.01,
    random_state=42
)
clf_best.fit(X_train_full, y_train_full)
```

 Python

9. Avaliação Final no Conjunto de Teste (20%)

Aplicamos o modelo treinado em X_{test} (30 exemplos) e obtivemos:

- **Acurácia no teste:** 1.000 (100%).
- **Métricas gerais:**

Classes	Precisão	Revocação	F1-Score	Suporte
Setosa	1.00	1.00	1.00	1.00
Versicolor	1.00	1.00	1.00	10
Virginica	1.00	1.00	1.00	10
Acurácia			1.00	30
Média	1.00	1.00	1.00	30
Média Ponderada	1.00	1.00	1.00	30

Tabela 7: Classification Report.

- **Matriz de Confusão:**

Classes	Setosa	Versicolor	Virginica
Setosa	10	0	0
Versicolor	0	10	0
Virginica	0	0	10

Tabela 8: Matriz de Confusão.

O modelo final classificou corretamente todas as instâncias do conjunto de testes.

10. Conclusões finais

1. A combinação `relu + hidden_layer_sizes=(50, 50) + learning_rate_init=0.01` apresentou a melhor média de validação (0,975) e foi escolhida como modelo final.
2. Apesar de `tanh` ter obtido bom desempenho no experimento inicial (0.967), `relu` conseguiu superar após a taxa de aprendizado.
3. A regra geral de adicionar profundidade (duas camadas ocultas) e aumentar o *learning rate* funcionou bem para `relu`, acelerando a convergência e melhorando a generalização no Iris.
4. Em outros problemas, deve-se testar também parâmetros como α , `early_stopping=True` e diversos solvers (`adam`), especialmente em datasets maiores e menos separáveis.
5. Os resultados mostram que, mesmo com dataset pequeno, arquiteturas profundas podem ter vantagem se combinadas com hiperparâmetros adequados.

11. Código

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

iris = load_iris()
X = iris.data
y = iris.target


X_train_full, X_test, y_train_full, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

activations = ['logistic', 'tanh', 'relu']
solver = 'sgd'
hidden_layers_initial = (100,)
results_initial = {}

for act in activations:
    train_scores = []
    val_scores = []
    loss_curves = []
    fold_idx = 1
    for train_idx, val_idx in skf.split(X_train_full, y_train_full):
        X_train, X_val = X_train_full[train_idx], X_train_full[val_idx]
        y_train, y_val = y_train_full[train_idx], y_train_full[val_idx]

        clf = MLPClassifier(
```

 Python


```

        activation=act,
        solver=solver,
        hidden_layer_sizes=hidden_layers_initial,
        random_state=42
    )
    clf.fit(X_train, y_train)

    y_pred_train = clf.predict(X_train)
    y_pred_val = clf.predict(X_val)
    train_acc = accuracy_score(y_train, y_pred_train)
    val_acc = accuracy_score(y_val, y_pred_val)

    train_scores.append(train_acc)
    val_scores.append(val_acc)
    loss_curves.append(clf.loss_curve_)
    fold_idx += 1

results_initial[act] = {
    'train_scores': train_scores,
    'val_scores': val_scores,
    'loss_curves': loss_curves
}

for act, res in results_initial.items():
    print(f"Activation: {act}")
    for i, (tr, vc) in enumerate(zip(res['train_scores'], res['val_scores']),
                                start=1):
        print(f"  Fold {i}: Train Acc = {tr:.3f}, Val Acc = {vc:.3f}")
    print(f"  Média Train Acc: {np.mean(res['train_scores']):.3f}, Média Val Acc: {np.mean(res['val_scores']):.3f}\n")

plt.figure(figsize=(10, 6))
for act, res in results_initial.items():
    for idx, lc in enumerate(res['loss_curves'], start=1):
        plt.plot(lc, label=f"{act} - Fold {idx}")
plt.title("Curvas de Perda - Parâmetros Iniciais")
plt.xlabel("Iterações")
plt.ylabel("Loss")
plt.legend(loc='upper right', fontsize='small', ncol=2)
plt.tight_layout()
plt.show()

hidden_layers_altered = (50, 50)
learning_rate_init_altered = 0.01
results_altered = {}

```

```
for act in activations:
    train_scores = []
    val_scores = []
    loss_curves = []
    for train_idx, val_idx in skf.split(X_train_full, y_train_full):
        X_train, X_val = X_train_full[train_idx], X_train_full[val_idx]
        y_train, y_val = y_train_full[train_idx], y_train_full[val_idx]

        clf = MLPClassifier(
            activation=act,
            solver=solver,
            hidden_layer_sizes=hidden_layers_altered,
            learning_rate_init=learning_rate_init_altered,
            random_state=42
        )
        clf.fit(X_train, y_train)

        y_pred_train = clf.predict(X_train)
        y_pred_val = clf.predict(X_val)
        train_acc = accuracy_score(y_train, y_pred_train)
        val_acc = accuracy_score(y_val, y_pred_val)

        train_scores.append(train_acc)
        val_scores.append(val_acc)
        loss_curves.append(clf.loss_curve_)

    results_altered[act] = {
        'train_scores': train_scores,
        'val_scores': val_scores,
        'loss_curves': loss_curves
    }

for act, res in results_altered.items():
    print(f"Activation (Altered): {act}")
    for i, (tr, vc) in enumerate(zip(res['train_scores'], res['val_scores']),
                                start=1):
        print(f"  Fold {i}: Train Acc = {tr:.3f}, Val Acc = {vc:.3f}")
    print(f"  Média Train Acc: {np.mean(res['train_scores']):.3f}, Média Val Acc: {np.mean(res['val_scores']):.3f}\n")

plt.figure(figsize=(10, 6))
for act, res in results_altered.items():
    for idx, lc in enumerate(res['loss_curves'], start=1):
        plt.plot(lc, label=f"{act} - Fold {idx}")
plt.title("Curvas de Perda - Parâmetros Alterados")
plt.xlabel("Iterações")
```

```

plt.ylabel("Loss")
plt.legend(loc='upper right', fontsize='small', ncol=2)
plt.tight_layout()
plt.show()

mean_vals_initial = {act: np.mean(res['val_scores']) for act, res in
results_initial.items()}
mean_vals_altered = {act: np.mean(res['val_scores']) for act, res in
results_altered.items()}

best_act_initial = max(mean_vals_initial, key=mean_vals_initial.get)
best_act_altered = max(mean_vals_altered, key=mean_vals_altered.get)

best_initial_score = mean_vals_initial[best_act_initial]
best_altered_score = mean_vals_altered[best_act_altered]

if best_altered_score >= best_initial_score:
    best_activation = best_act_altered
    best_params = {
        'hidden_layer_sizes': hidden_layers_altered,
        'learning_rate_init': learning_rate_init_altered,
        'activation': best_act_altered,
        'solver': solver
    }
    print(f"Melhor modelo: Ativação={best_act_altered} (Alterado), Val Acc
Média={best_altered_score:.3f}")
else:
    best_activation = best_act_initial
    best_params = {
        'hidden_layer_sizes': hidden_layers_initial,
        'activation': best_act_initial,
        'solver': solver
    }
    print(f"Melhor modelo: Ativação={best_act_initial} (Inicial), Val Acc
Média={best_initial_score:.3f}")

clf_best = MLPClassifier(
    activation=best_activation,
    solver=solver,
    hidden_layer_sizes=best_params['hidden_layer_sizes'],
    **({'learning_rate_init': best_params['learning_rate_init']} if
'learning_rate_init' in best_params else {}),
    random_state=42
)
clf_best.fit(X_train_full, y_train_full)

y_pred_test = clf_best.predict(X_test)

```

```
test_acc = accuracy_score(y_test, y_pred_test)
print(f"\nAcurácia no conjunto de teste: {test_acc:.3f}\n")
print("Classification Report:")
print(classification_report(y_test, y_pred_test, target_names=iris.target_names))
print("Matriz de Confusão:")
print(confusion_matrix(y_test, y_pred_test))
```