Entendem os codigos a seguir ..... genericos e depois ajustem para o caso de voces.

2. Cálculo para Cada Variável

(a) `numel(W1)`

- `W1` : Matriz de pesos da **camada de entrada para a oculta**.
  - Dimensão: `[20 x 2]` (20 neurônios, 2 entradas cada).
  - Número de elementos:

$$20 \times 2 = 40 \text{ pesos}$$

(b) `numel(b1)`

- `b1` : Vetor de vieses da **camada oculta**.
  - Dimensão: `[20 x 1]` (1 viés por neurônio oculto).
  - Número de elementos:

$$20 \times 1 = 20 \text{ vieses}$$

(c) `numel(W2)`

- `W2` : Matriz de pesos da **camada oculta para a saída**.
  - Dimensão: `[1 x 20]` (1 neurônio de saída, 20 conexões).
  - Número de elementos:

$$1 \times 20 = 20 \text{ pesos}$$

(d) `numel(b2)`

- `b2` : Viés da **camada de saída**.
  - Dimensão: `[1 x 1]` (1 viés para o neurônio de saída).
  - Número de elementos:

$$1 \times 1 = 1 \text{ vi}$$
s

3. Soma Total

Total de par
metros = + + + = 81

4. Verificação Prática no MATLAB

Execute este código para confirmar:

matlab Copy Download

```matlab
% Inicialização dos pesos (exemplo)
W1 = randn(20, 2);  % 20x2
b1 = zeros(20, 1);   % 20x1
W2 = randn(1, 20);   % 1x20
b2 = 0;              % 1x1

% Cálculo do total de parâmetros
total_params = numel(W1) + numel(b1) + numel(W2) + numel(b2);
fprintf('Total de parâmetros: %d\n', total_params);
```

Saída:

text Copy Download

```
Total de parâmetros: 81
```

```matlab
load('melhores_pesos.mat');
% Dados novos (previamente normalizados)
Z1 = W1 * X_novo + b1;
A1 = 1 ./ (1 + exp(-Z1));
Z2 = W2 * A1 + b2;
previsao = Z2 * std(Y) + mean(Y); % Desnormaliza a saída
```

Segue codigo referencia usado em anos anteriores, independente de versao Matlab. ja entenderam o pipeline, gradiente/derivadas das funcoes:

```
%% Carregar os dados do arquivo .mat
load('dados.mat'); % Arquivo contendo Input1, Input2, Output

% Preparar os dados de entrada e saída
X = [Input1, Input2]'; % Entradas (2xN)
Y = Output'; % Saída (1xN)

%% Definir a arquitetura da rede neural
input_size = 2; % 2 entradas
hidden_size = 20; % 20 neurônios na camada oculta
output_size = 1; % 1 saída

% Inicialização dos pesos (aleatórios)
W1 = randn(hidden_size, input_size) * 0.1; % Pesos entrada -> oculta
b1 = zeros(hidden_size, 1); % Bias camada oculta
W2 = randn(output_size, hidden_size) * 0.1; % Pesos oculta -> saída
b2 = zeros(output_size, 1); % Bias camada saída

%% Parâmetros de treinamento
learning_rate = 0.1; % Taxa de aprendizado
epochs = 1000; % Número de épocas
loss_history = zeros(1, epochs); % Armazenar perda ao longo do treino

%% Treinamento (Gradiente Descendente + Backpropagation)
for epoch = 1:epochs
    % Forward Propagation
    Z1 = W1 * X + b1; % Saída camada oculta (antes da ativação)
    A1 = 1 ./ (1 + exp(-Z1)); % Ativação sigmoide (logsig)
    Z2 = W2 * A1 + b2; % Saída camada saída (antes da ativação)
    A2 = Z2; % Saída linear (sem ativação, pois é regressão)

    % Cálculo do erro (MSE - Mean Squared Error)
    loss = mean((A2 - Y).^2);
    loss_history(epoch) = loss;

    % Backpropagation
    dA2 = 2 * (A2 - Y) / size(X, 2); % Gradiente da perda em relação a A2
    dZ2 = dA2; % Para saída linear, derivada é 1

    dW2 = dZ2 * A1'; % Gradiente em relação a W2
    db2 = sum(dZ2, 2); % Gradiente em relação a b2

    dA1 = W2' * dZ2; % Gradiente em relação a A1
    dZ1 = dA1 .* (A1 .* (1 - A1)); % Gradiente em relação a Z1 (sigmoide)

    dW1 = dZ1 * X'; % Gradiente em relação a W1
    db1 = sum(dZ1, 2); % Gradiente em relação a b1

    % Atualização dos pesos (Gradiente Descendente)
    W1 = W1 - learning_rate * dW1;
    b1 = b1 - learning_rate * db1;
    W2 = W2 - learning_rate * dW2;
    b2 = b2 - learning_rate * db2;

    % Exibir progresso a cada 100 épocas
    if mod(epoch, 100) == 0
        fprintf('Época %d, Perda: %.4f\n', epoch, loss);
    end
end

%% Salvar os pesos em um arquivo pesos.mat
save('pesos.mat', 'W1', 'b1', 'W2', 'b2');
disp('Pesos salvos em pesos.mat');

%% Plotar a perda ao longo do treinamento
figure;
plot(1:epochs, loss_history);
title('Perda durante o Treinamento');
xlabel('Época');
ylabel('MSE');

%% Testar a rede neural (exemplo com os mesmos dados)
Z1 = W1 * X + b1;
A1 = 1 ./ (1 + exp(-Z1));
Z2 = W2 * A1 + b2;
```

```
A2 = Z2;

% Plotar saída real vs. saída prevista
figure;
plot(Y, 'b', 'LineWidth', 2); hold on;
plot(A2, 'r--', 'LineWidth', 1.5);
legend('Saída Real', 'Saída Prevista');
title('Comparação: Saída Real vs. Saída Prevista');
xlabel('Amostra');
ylabel('Valor');
```

Outra opcao pipeline com (1.2) Normalizar os Dados

1. Pré-Processamento dos Dados

(1.1) Carregar Dados

matlab Copy Download

```matlab
load('dados.mat'); % Arquivo com Input1, Input2, Output (séries temporais)
```

(1.2) Normalizar os Dados

matlab Copy Download

```matlab
X = [Input1; Input2]; % Matriz [2 x N]
Y = Output;           % Vetor [1 x N]

% Normalização (Z-score)
X = (X - mean(X, 2)) ./ std(X, 0, 2);
Y = (Y - mean(Y)) / std(Y);
```

(1.3) Divisão Treino/Teste (Preservando Ordem Temporal)

matlab Copy Download

```matlab
train_ratio = 0.7; % 70% treino, 30% teste
N_train = floor(N * train_ratio);

X_train = X(:, 1:N_train);
Y_train = Y(:, 1:N_train);

X_test = X(:, N_train+1:end);
Y_test = Y(:, N_train+1:end);
```

2. Configuração da Rede Neural

(2.1) Parâmetros da Arquitetura

matlab Copy Download

```matlab
input_size = 2;    % 2 entradas (Input1, Input2)
hidden_size = 20;  % 20 neurônios na camada oculta
output_size = 1;   % 1 saída (Output)

learning_rate = 0.1; % Taxa de aprendizado
epochs = 1000;       % Número de épocas
```

(2.2) Inicialização dos Pesos

matlab Copy Download

```matlab
% Pesos e biases aleatórios (pequenos valores)
W1 = randn(hidden_size, input_size) * 0.1;
b1 = zeros(hidden_size, 1);

W2 = randn(output_size, hidden_size) * 0.1;
b2 = zeros(output_size, 1);
```

3. Treinamento da Rede

(3.1) Loop de Treinamento (Forward + Backpropagation)

matlab Copy Download

```matlab
loss_history = zeros(1, epochs); % Armazenar perda

for epoch = 1:epochs
    % Forward Pass
    Z1 = W1 * X_train + b1;
    A1 = 1 ./ (1 + exp(-Z1)); % Sigmoid
    Z2 = W2 * A1 + b2;
    A2 = Z2; % Saída linear (regressão)

    % Cálculo da Perda (MSE)
    loss = mean((A2 - Y_train).^2);
    loss_history(epoch) = loss;

    % Backpropagation
```

```
    dA2 = 2 * (A2 - Y_train) / size(X_train, 2);
    dZ2 = dA2;

    dW2 = dZ2 * A1';
    db2 = sum(dZ2, 2);

    dA1 = W2' * dZ2;
    dZ1 = dA1 .* (A1 .* (1 - A1));

    dW1 = dZ1 * X_train';
    db1 = sum(dZ1, 2);

    % Atualização dos Pesos (Gradiente Descendente)
    W1 = W1 - learning_rate * dW1;
    b1 = b1 - learning_rate * db1;
    W2 = W2 - learning_rate * dW2;
    b2 = b2 - learning_rate * db2;

    % Log de Progresso
    if mod(epoch, 100) == 0
        fprintf('Época %d, Perda: %.4f\n', epoch, loss);
    end
end
```

(3.2) Salvamento dos Pesos Treinados

matlab Copy Download

```
save('pesos_treinados.mat', 'W1', 'b1', 'W2', 'b2');
```

4. Avaliação do Modelo

(4.1) Teste com Dados Não Vistos

matlab Copy Download

```
% Forward Pass nos dados de teste
Z1_test = W1 * X_test + b1;
A1_test = 1 ./ (1 + exp(-Z1_test));
Z2_test = W2 * A1_test + b2;
A2_test = Z2_test;

% Cálculo do MSE no teste
test_loss = mean((A2_test - Y_test).^2);
fprintf('Perda no Teste: %.4f\n', test_loss);
```

(4.2) Visualização dos Resultados

matlab Copy Download

```
figure;
plot(Y_test, 'b', 'LineWidth', 2); hold on;
plot(A2_test, 'r--', 'LineWidth', 1.5);
legend('Real', 'Previsto');
title('Comparação: Valores Reais vs. Previstos');
xlabel('Amostra');
ylabel('Valor Normalizado');

figure;
plot(loss_history);
title('Evolução da Perda durante o Treinamento');
xlabel('Época');
ylabel('MSE');
```

5. Pipeline Resumido

1. **Pré-processamento**:
   - Carregar dados → Normalizar → Dividir (treino/teste).
2. **Configuração da Rede**:
   - Definir arquitetura (2-20-1) → Inicializar pesos.
3. **Treinamento**:
   - Forward pass → Backpropagation → Atualização dos pesos.
4. **Avaliação**:
   - Teste com dados não vistos → Métricas de erro → Visualização.

Observações Finais

- **Para Séries Temporais**: Se quiser usar **janelas temporais**, modifique `X` para incluir lags (ex.: `X = [x(t-1); x(t-2)]`).
- **Melhorias**: Adicione **early stopping** ou **validação cruzada temporal** para evitar overfitting.

Pipeline Avançado com Early Stopping e Validação Temporal

Melhorias adicionadas:
✅ **Early Stopping**: Interrompe o treinamento se a perda no conjunto de validação não melhorar após um número definido de épocas (`patience`).

✅ **Validação Temporal**: Divide os dados sequencialmente (sem aleatoriedade) para preservar a estrutura temporal.

1. Pré-Processamento dos Dados

matlab Copy Download

```matlab
% Carrega os dados da série temporal
load('dados.mat'); % Arquivo com Input1, Input2, Output

% Normalização (Z-score para melhor convergência)
X = [Input1; Input2];
Y = Output;
X = (X - mean(X, 2)) ./ std(X, 0, 2); % Normaliza cada feature
Y = (Y - mean(Y)) / std(Y); % Normaliza a saída

% Divisão Treino/Validação/Teste (60%/20%/20%)
N = size(X, 2);
train_ratio = 0.6; val_ratio = 0.2;
N_train = floor(N * train_ratio);
N_val = floor(N * val_ratio);

X_train = X(:, 1:N_train);
Y_train = Y(:, 1:N_train);
X_val = X(:, N_train+1:N_train+N_val);
Y_val = Y(:, N_train+1:N_train+N_val);
X_test = X(:, N_train+N_val+1:end);
Y_test = Y(:, N_train+N_val+1:end);
```

2. Configuração da Rede Neural com Early Stopping

matlab Copy Download

```matlab
% Arquitetura da Rede
input_size = 2;      % 2 entradas (Input1, Input2)
hidden_size = 20;    % 20 neurônios na camada oculta
output_size = 1;     % 1 saída (regressão)

% Hiperparâmetros
learning_rate = 0.1;
epochs = 1000;
patience = 50;       % Número de épocas sem melhoria para parar
min_delta = 1e-4;    % Melhoria mínima para ser considerada

% Inicialização dos pesos
W1 = randn(hidden_size, input_size) * 0.1;
b1 = zeros(hidden_size, 1);
W2 = randn(output_size, hidden_size) * 0.1;
b2 = zeros(output_size, 1);

% Variáveis para Early Stopping
best_val_loss = Inf;
wait = 0; % Contador de épocas sem melhoria
```

3. Treinamento com Validação e Early Stopping

matlab Copy Download

```matlab
loss_history = zeros(1, epochs);
val_loss_history = zeros(1, epochs);

for epoch = 1:epochs
    % Forward Pass (Treino)
    Z1 = W1 * X_train + b1;
    A1 = 1 ./ (1 + exp(-Z1)); % Sigmoid
    Z2 = W2 * A1 + b2;
    A2 = Z2;

    % Cálculo da Perda (MSE)
    train_loss = mean((A2 - Y_train).^2);
    loss_history(epoch) = train_loss;

    % Forward Pass (Validação)
    Z1_val = W1 * X_val + b1;
    A1_val = 1 ./ (1 + exp(-Z1_val));
    Z2_val = W2 * A1_val + b2;
    A2_val = Z2_val;
    val_loss = mean((A2_val - Y_val).^2);
    val_loss_history(epoch) = val_loss;

    % Backpropagation (Treino)
    dA2 = 2 * (A2 - Y_train) / size(X_train, 2);
    dZ2 = dA2;
    dW2 = dZ2 * A1';
    db2 = sum(dZ2, 2);
    dA1 = W2' * dZ2;
    dZ1 = dA1 .* (A1 .* (1 - A1));
    dW1 = dZ1 * X_train';
    db1 = sum(dZ1, 2);

    % Atualização dos Pesos
    W1 = W1 - learning_rate * dW1;
    b1 = b1 - learning_rate * db1;
    W2 = W2 - learning_rate * dW2;
    b2 = b2 - learning_rate * db2;

    % Early Stopping
```

```
    if val_loss < best_val_loss - min_delta
        best_val_loss = val_loss;
        wait = 0;
        % Salva os melhores pesos
        best_W1 = W1; best_b1 = b1;
        best_W2 = W2; best_b2 = b2;
    else
        wait = wait + 1;
        if wait >= patience
            fprintf('Early Stopping na época %d\n', epoch);
            break;
        end
    end

    % Log de Progresso
    if mod(epoch, 50) == 0
        fprintf('Época %d: Perda Treino = %.4f | Perda Validação = %.4f\n', epoch, train_loss, val_loss);
    end
end

% Restaura os melhores pesos encontrados
W1 = best_W1; b1 = best_b1;
W2 = best_W2; b2 = best_b2;
```

4. Avaliação Final e Visualização

matlab Copy Download

```
% Teste com os melhores pesos
Z1_test = W1 * X_test + b1;
A1_test = 1 ./ (1 + exp(-Z1_test));
Z2_test = W2 * A1_test + b2;
A2_test = Z2_test;
test_loss = mean((A2_test - Y_test).^2);
fprintf('Perda no Teste: %.4f\n', test_loss);

% Gráficos
figure;
subplot(2,1,1);
plot(loss_history(1:epoch), 'b', 'LineWidth', 1.5); hold on;
plot(val_loss_history(1:epoch), 'r', 'LineWidth', 1.5);
legend('Treino', 'Validação');
title('Evolução da Perda');
xlabel('Época'); ylabel('MSE');

subplot(2,1,2);
plot(Y_test, 'b', 'LineWidth', 2); hold on;
plot(A2_test, 'r--', 'LineWidth', 1.5);
legend('Real', 'Previsto');
title('Previsões no Conjunto de Teste');
xlabel('Amostra'); ylabel('Valor Normalizado');

% Salva os pesos finais
save('melhores_pesos.mat', 'W1', 'b1', 'W2', 'b2');
```

Explicação das Melhorias

1. **Validação Temporal**:
   - Os dados são divididos em **treino (60%)**, **validação (20%)**, e **teste (20%)**, **preservando a ordem temporal**.
   - A validação é usada para monitorar overfitting.
2. **Early Stopping**:
   - Se a perda na validação não melhorar por `patience` épocas, o treinamento é interrompido.
   - Os **melhores pesos** são salvos quando a perda de validação atinge um mínimo.
3. **Visualização**:
   - Gráficos comparativos de perda (treino/validação) e previsões (real vs. previsto).

Como Usar em Produção

matlab Copy Download

```
load('melhores_pesos.mat');
% Dados novos (previamente normalizados)
Z1 = W1 * X_novo + b1;
A1 = 1 ./ (1 + exp(-Z1));
Z2 = W2 * A1 + b2;
previsao = Z2 * std(Y) + mean(Y); % Desnormaliza a saída
```

Este código está pronto para **séries temporais** com mitigação de overfitting!

Explicacoes abaixo. Entender a Rede neural e o pipeline:

1. **Carregamento dos Dados**:
   - Assume-se que `dados.mat` contém `Input1`, `Input2` e `Output`.
   - Os dados são organizados em matrizes `X` (entradas) e `Y` (saída).
2. **Inicialização da Rede**:
   - Pesos (`W1`, `W2`) e vieses (`b1`, `b2`) são inicializados aleatoriamente.
   - A camada oculta usa **20 neurônios com ativação sigmoide (`logsig`)**.
   - A camada de saída é **linear** (sem ativação, pois é um problema de regressão).

3. **Treinamento (Gradiente Descendente + Backpropagation)**:
    - **Forward Pass**: Calcula a saída da rede.
    - **Cálculo da Perda (MSE)**: Mede o erro entre a saída prevista e a real.
    - **Backward Pass**: Calcula os gradientes e ajusta os pesos.
4. **Visualização**:
    - Gráfico da perda ao longo das épocas.
    - Comparação entre saída real e prevista.

Resumo das Derivadas

Derivada	Fórmula Matemática	Dimensões (MATLAB)	Explicação
`dA2`	$\frac{(A2 - Y)}{N}$	(1xN)	Gradiente da perda em relação à saída.
`dZ2`	`dA2`	(1xN)	$A2 = Z2$, a derivada é 1.
`dW2`	`dZ2 * A1'`	(1x20)	Gradiente em relação aos pesos da saída.
`db2`	`sum(dZ2, 2)`	(1x1)	Gradiente em relação ao bias da saída.
`dA1`	`W2' * dZ2`	(20xN)	Gradiente em relação à ativação da oculta.
`dZ1`	`dA1 .* (A1 .* (1 - A1))`	(20xN)	Gradiente em relação a `Z1` (sigmoide).
`dW1`	`dZ1 * X'`	(20x2)	Gradiente em relação aos pesos da oculta.
`db1`	`sum(dZ1, 2)`	(20x1)	Gradiente em relação ao bias da oculta.

4. Por Que Usamos Essas Derivadas?

- **Backpropagation** propaga o erro da saída para trás, ajustando os pesos para minimizar a perda.
- Cada derivada indica **como pequenas mudanças nos pesos afetam a perda total**.
- **Gradiente Descendente** usa essas derivadas para atualizar os pesos:

$$W = W - \eta \cdot$$

Os **4 vetores/matrizes de pesos (W1, W2, b1, b2)** existem porque sua rede neural possui:

1. **Uma camada oculta** (com 20 neurônios).
2. **Uma camada de saída** (com 1 neurônio).

Cada camada tem seus próprios **pesos (W)** e **vieses (b)**, que são aprendidos durante o treinamento. Vamos decompor:

1. Estrutura da Rede Neural

- **Entrada**: 2 features (`Input1` , `Input2`) → **2 neurônios de entrada**.
- **Camada Oculta**: 20 neurônios → **precisa de pesos e vieses para conectar a entrada**.
- **Saída**: 1 neurônio → **precisa de pesos e vieses para conectar a camada oculta**.

2. Por Que 4 Vetores/Matrizes?

Parâmetro	Dimensão	Explicação
W1	`[20 x 2]`	Pesos que conectam a **entrada (2 features)** à **camada oculta (20 neurônios)**.
b1	`[20 x 1]`	Bias (viés) adicionado a cada neurônio da **camada oculta** (20 termos, um por neurônio).
W2	`[1 x 20]`	Pesos que conectam a **camada oculta (20 neurônios)** à **saída (1 neurônio)**.
b2	`[1 x 1]`	Bias (viés) do neurônio de **saída**.

3. Ilustração Matemática

(a) Forward Pass (Cálculo das Saídas)

1. **Camada Oculta**:

$$Z1 = X \cdot W1 + b1 \quad \text{(20 neurônios)}$$

   - `W1` : Transforma 2 entradas em 20 saídas.
   - `b1` : Desloca a ativação de cada neurônio.
2. **Camada de Saída**:

$$Z2 = A1 \cdot W2 + b2 \quad \text{(1 neurônio)}$$

   - `W2` : Combina as 20 ativações da camada oculta em 1 saída.
   - `b2` : Ajuste final da saída.

4. Por Que Não Podemos Ter Apenas W1 e W2?

- **Bias (`b1` e `b2`)** é essencial para:
  - Permitir que a rede aprenda **deslocamentos** (offsets) nos dados.
  - Garantir que os neurônios possam ativar mesmo quando a entrada é zero.
  - Exemplo: Sem `b1` , se `X = [0, 0]` , então `Z1 = W1 * [0,0] = 0` → a rede perde flexibilidade.

5. Analogia com Equação Linear

Pense em uma regressão linear simples:

$$y = x \cdot w + b$$

- `w` = peso (controla a inclinação).
- `b` = bias (controla o intercepto).

Na rede neural:

- Cada neurônio tem seu próprio `w` e `b` para capturar relações não-lineares complexas.

6. Visualização Gráfica

text Copy Download

```
CAMADA DE ENTRADA       CAMADA OCULTA (20)       CAMADA DE SAÍDA
[Input1] → (W1, b1) → Neurônio 1  → (W2, b2) → [Output]
[Input2] → (W1, b1) → Neurônio 2  → (W2, b2) → [Output]
                  ... → Neurônio 20 → (W2, b2) → [Output]
```

- **W1**: Mapeia 2 entradas → 20 neurônios ocultos.
- **b1**: Viés para cada um dos 20 neurônios.
- **W2**: Mapeia 20 neurônios ocultos → 1 saída.
- **b2**: Viés da saída.

7. Número Total de Parâmetros

Para sua rede:

- **W1**: `20 x 2 = 40` pesos.
- **b1**: `20 x 1 = 20` vieses.
- **W2**: `1 x 20 = 20` pesos.

- **b2**: `1 x 1 = 1` viés.

$$1 + 40 + 20 + 20 = 81$$

Total: 81 parâmetros treináveis.

8. Conclusão

Os **4 vetores (W1, b1, W2, b2)** são necessários porque:

1. Cada camada (oculta e saída) precisa de seus próprios **pesos** para transformar os dados.
2. Os **vieses** garantem que a rede possa modelar comportamentos mesmo com entradas zeradas.
3. Sem `b1` ou `b2`, a rede perderia capacidade de aprendizado (como uma regressão linear sem intercepto).

Esses parâmetros são ajustados durante o **backpropagation** para minimizar o erro!

Em uma rede neural, **`z1` e `z2`** representam os **sinais brutos** (não ativados) calculados antes da aplicação da função de ativação em cada camada. Eles são resultados da combinação linear entre:

- **Entradas** (ou saídas da camada anterior).
- **Pesos** (`W1`, `W2`).
- **Vieses** (`b1`, `b2`).

1. Definição Matemática

(a) `z1` : Sinal Bruto da Camada Oculta

$$Z1 = W1 \cdot X + b1$$

- **`W1`** : Matriz de pesos da camada de entrada para a oculta (dimensão `[20 x 2]`).
- **`x`** : Dados de entrada (dimensão `[2 x N]`, onde `N` = número de amostras).
- **`b1`** : Vetor de vieses da camada oculta (`[20 x 1]`).
- **Saída**: `Z1` tem dimensão `[20 x N]` (20 neurônios × N amostras).

O que `z1` representa?

- É a **combinação linear** das entradas antes de passar pela função de ativação (sigmoide, no seu caso).
- Cada linha de `Z1` corresponde ao sinal de **um neurônio oculto** para todas as amostras.

(b) `z2` : Sinal Bruto da Camada de Saída

$$Z2 = W2 \cdot A1 + b2$$

- **`W2`** : Matriz de pesos da camada oculta para a saída (`[1 x 20]`).
- **`A1`** : Saída ativada da camada oculta (`A1 = sigmoide(Z1)` , dimensão `[20 x N]`).
- **`b2`** : Viés da camada de saída (`[1 x 1]`).
- **Saída**: `Z2` tem dimensão `[1 x N]` (1 neurônio de saída × N amostras).

O que `z2` representa?

- É a **combinação linear** das ativações da camada oculta (`A1`) antes de qualquer ajuste final.
- No seu caso, a saída é linear (`A2 = Z2`), mas em problemas de classificação, poderia passar por outra ativação (ex.: sigmoide para probabilidades).

2. Relação com os Pesos (`W1` , `W2` , `b1` , `b2`)

Os pesos são usados para calcular `Z1` e `Z2` :

1. **`W1` e `b1`** definem como as **entradas** são transformadas nos sinais ocultos (`Z1`).
2. **`W2` e `b2`** definem como as **ativações ocultas (`A1`)** são transformadas na saída bruta (`Z2`).

Exemplo Numérico Simplificado

Suponha:

- Entrada: `X = [1.0, 0.5]^T` (2 features).
- `W1` (um neurônio oculto): `[0.4, -0.1]`, `b1 = 0.2`.
- Cálculo de `Z1`:

$$Z1 = 0.4 \cdot 1.0 + (-0.1) \cdot 0.5 + 0.2 = 0.4 - 0.05 + 0.2 = 0.55$$

- Se usarmos sigmoide: `A1 = 1 / (1 + e^{-0.55}) ≈ 0.63`.

3. Papel no Backpropagation

Durante o treinamento:

1. **Forward Pass**:
   - Calculamos `Z1`, `A1`, `Z2`, `A2`.
2. **Backward Pass**:
   - Derivamos a perda em relação a `Z2` (`dZ2`), depois em relação a `Z1` (`dZ1`).
   - Esses gradientes são usados para atualizar `W1`, `W2`, `b1`, `b2`.

Por que `Z1` e `Z2` são importantes?

- São os **pontos onde os pesos são aplicados** antes da não-linearidade (ativação).
- As derivadas (`dZ1`, `dZ2`) indicam **como ajustar os pesos** para reduzir o erro.

4. Comparação com `A1` e `A2`

Variável	Fórmula	Papel	Dimensão (Exemplo)
`Z1`	`W1 * X + b1`	Sinal bruto da camada oculta.	`[20 x N]`
`A1`	`sigmoide(Z1)`	Saída ativada da camada oculta.	`[20 x N]`
`Z2`	`W2 * A1 + b2`	Sinal bruto da camada de saída.	`[1 x N]`
`A2`	`Z2` (saída linear)	Saída final da rede.	`[1 x N]`

5. Resumo Visual

text Copy Download

```
INPUT (X) → [Z1 = W1*X + b1] → [A1 = sigmoide(Z1)] → [Z2 = W2*A1 + b2] → OUTPUT (A2 = Z2)
```

- **`Z1` e `Z2`** são estágios intermediários **antes** da ativação.
- **`A1` e `A2`** são resultados **após** a ativação (ou saída linear).

Conclusão

- **`Z1`** e **`Z2`** são as **transformações lineares** cruciais para o cálculo das saídas da rede.
- Eles dependem diretamente dos pesos (`W1`, `W2`) e vieses (`b1`, `b2`), que são ajustados durante o treinamento.
- Sem `Z1` e `Z2`, não haveria como propagar os gradientes no backpropagation!

1. O que é `N` no Código?

- **`size(X, 2)`** retorna o número de colunas de `X`, que corresponde a `N` (quantidade de dados da série temporal).
- Se sua série temporal tem, por exemplo, **1000 pontos**, então `N = 1000`.

2. Como os Dados são Organizados?

No MATLAB, seu arquivo `.mat` deve ter esta estrutura:

Variável	Dimensão	Exemplo (N=1000)	Descrição
`Input1`	`[1 x N]`	`[1 x 1000]`	Valores da entrada 1 ao longo do tempo.
`Input2`	`[1 x N]`	`[1 x 1000]`	Valores da entrada 2 ao longo do tempo.
`Output`	`[1 x N]`	`[1 x 1000]`	Valores reais da saída (target).

Ao concatenar as entradas:

matlab Copy Download

```matlab
X = [Input1; Input2]; % Matriz [2 x N]
Y = Output;           % Vetor [1 x N]
```

3. Impacto de `N` no Cálculo das Derivadas

As derivadas (`dW1` , `dW2` , `db1` , `db2`) são **médias dos gradientes** sobre todas as amostras (`N`). Por exemplo:

(a) Gradiente da Perda (`dA2`)

$$dA2(A2 = \frac{1}{N} Y)$$

- Quanto maior `N` , menor o impacto de cada amostra individual no gradiente (evita overfitting).

(b) Gradiente dos Vieses (`db1` , `db2`)

matlab Copy Download

```matlab
db2 = sum(dZ2, 2); % Soma os gradientes ao longo de N amostras
```

- Como `dZ2` tem dimensão `[1 x N]` , `sum(dZ2, 2)` soma os erros de todas as amostras.

4. Adaptação para Séries Temporais

Se você está modelando uma série temporal:

1. `X` pode ser uma **janela deslizante** dos dados passados (ex.: usar `Input1(t-1)` , `Input1(t-2)` como entradas).
2. `Y` é o valor futuro a ser previsto (ex.: `Output(t)`).

Exemplo de Janelamento (Lag Features)

Para prever `y(t)` com base nos últimos 3 pontos:

matlab Copy Download

```matlab
Input1 = [x(1), x(2), ..., x(N-3)]; % t-3
Input2 = [x(2), x(3), ..., x(N-2)]; % t-2
Output  = [x(4), x(5), ..., x(N)];   % t
```

5. Cuidados com Séries Temporais

1. **Normalização**:
   - Normalize `Input1` , `Input2` , e `Output` para evitar disparidade de escalas.

   matlab Copy Download

   ```matlab
   X = (X - mean(X, 2)) ./ std(X, 0, 2);
   ```
2. **Divisão Treino/Teste**:
   - Não randomize! Use os primeiros 70% para treino e os 30% finais para teste.

   matlab Copy Download

   ```matlab
   train_ratio = 0.7;
   N_train = floor(N * train_ratio);
   X_train = X(:, 1:N_train);
   Y_train = Y(:, 1:N_train);
   ```

6. Resumo

- `N` : Número de pontos na série temporal.

- `X` : Matriz `[2 x N]` com as entradas.
- `Y` : Vetor `[1 x N]` com as saídas.
- **Gradientes**: São calculados considerando **todas as amostras** (`N`).

Em ter., 27 de mai. de 2025 às 16:20, Alberto Alvares <albertojalvares@alvarestech.com> escreveu:

> Danilo,
>
> Segue fluxo Node-RED esta no diretorio do Teams Microsoft UnB - Disciplinas: tente abrir
>
> Uso material do meu ex-aluno doutorado Efrain: https://www.youtube.com/@MonkeyWit
>
> Veja videos sobre Node-RED e como trabalhar com ele ... ja instalamos na ultima aula no seu computador: https://www.youtube.com/watch?v=bdyRjRO_7mw
>
> importe o fluxo que estou enviando e instale os node que vai precisar.
>
> Em dom., 25 de mai. de 2025 às 13:06, Alberto Alvares <albertojalvares@alvarestech.com> escreveu:
>
> > Boa tarde,
> >
> > Tentem desenvolver, adaptando o código Matlab, da Aula3, usando o gerador de simulação ICONICS conforme demonstrado no vídeo. Para desenvolver GUI Matlab aula3.m fou usado o GUIDE, Matlab até 2015. Após esta data, versão, outro ambiente de desenvolvimento de GUI é o App Designer.
> >
> > ICONICS Suite Demo I MEIDS Software Solutions
> >
> > Maquina virtual Windows XP Vmware: https://unbbr.sharepoint.com/:f:/s/AutomaodeProcessos/Eu4RRTzmBbZGt2fUTQymgywB7mg6uU4LxYRkIytk1tKCeg?e=raZdbQ
> >
> > Matlab 13: Matlab2013-iso
> >
> > Assirtir a Aula URL para entender melhor projeto controlador PID, sintese manual, importante para entender como projetar um PID:
> >
> > https://www.youtube.com/watch?v=r-kfkjUvPa0
> >
> > Sintonia PID metodos: PowerPoint Presentation
> >
> > https://www.mathworks.com/help/control/getstart/designing-pid-controllers-with-the-pid-tuner-gui.html
> >
> > Softwares para OPC-DA, Iconics, software Matlab anos anteriores incluindo Aula 3, codigo Matlab, entre outros.
> > SoftwaresProjetosRemotos
> >
> > Tentem instalar no windows de voces com Matlab, ferramenta Opctools para acessar servidores OPC local. Instalar o Iconics servidor OPC DA, usem o material da Aula 3, mostrei aula passada e substituam o nome das Tags por Tags Iconics.
> >
> > Qualquer duvida entrem em contato. Tentem fazer este exercicio, so assim ira entender.
> >
> > Dependendo PC de voces, instalem um vmware e importem a maquina virtual Windows XP para melhor compatibilidade com servidor OPC DA da planta SMAR.
> >
> > Importante exercicio aprender a usar os recursos para Automacao de Processos.
> >
> > --
> > Um abraco,
> >
> > Alberto J. Alvares
> > Prof. Titular

--
Um abraco,

Alberto J. Alvares
Prof. Titular