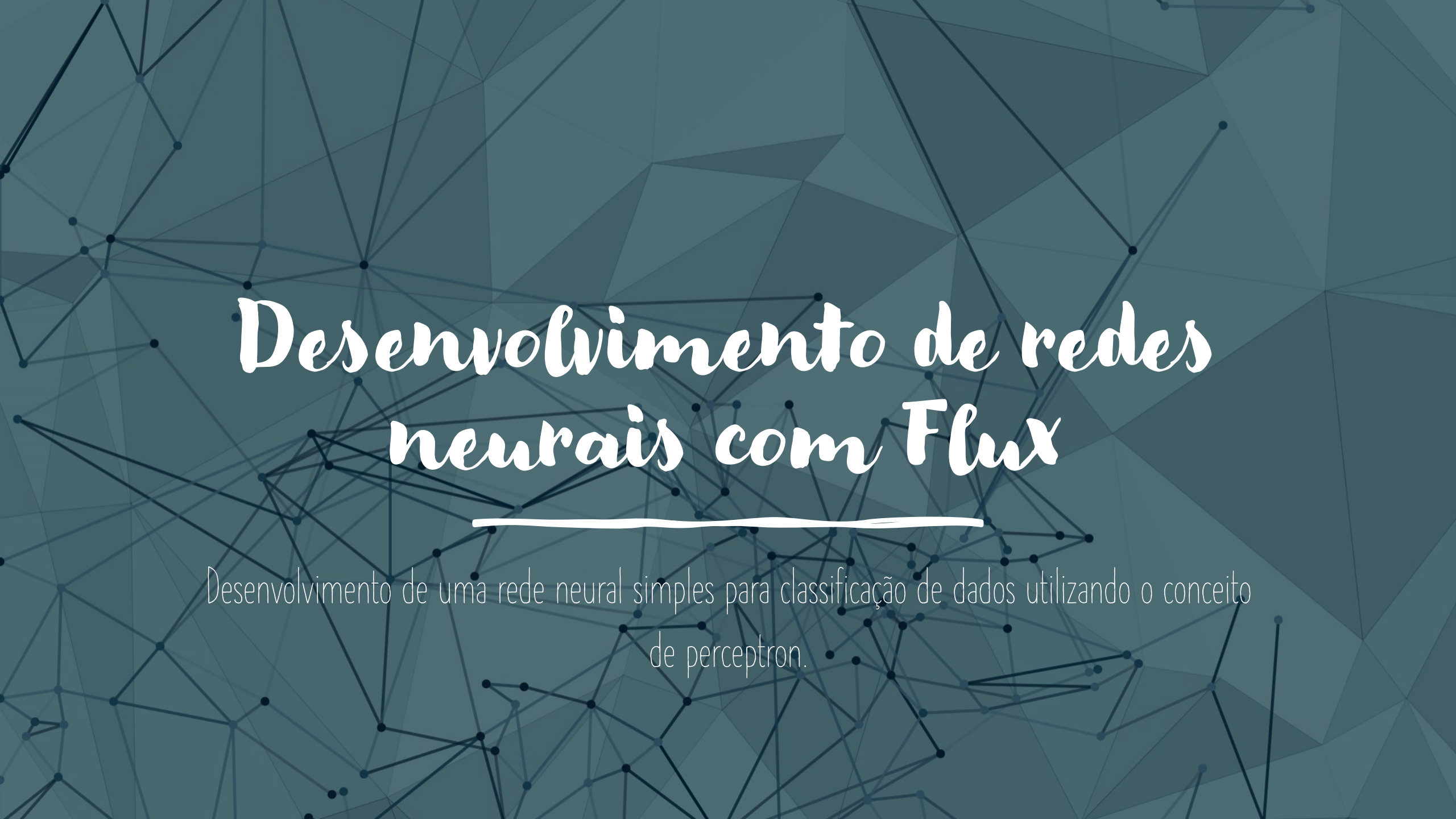
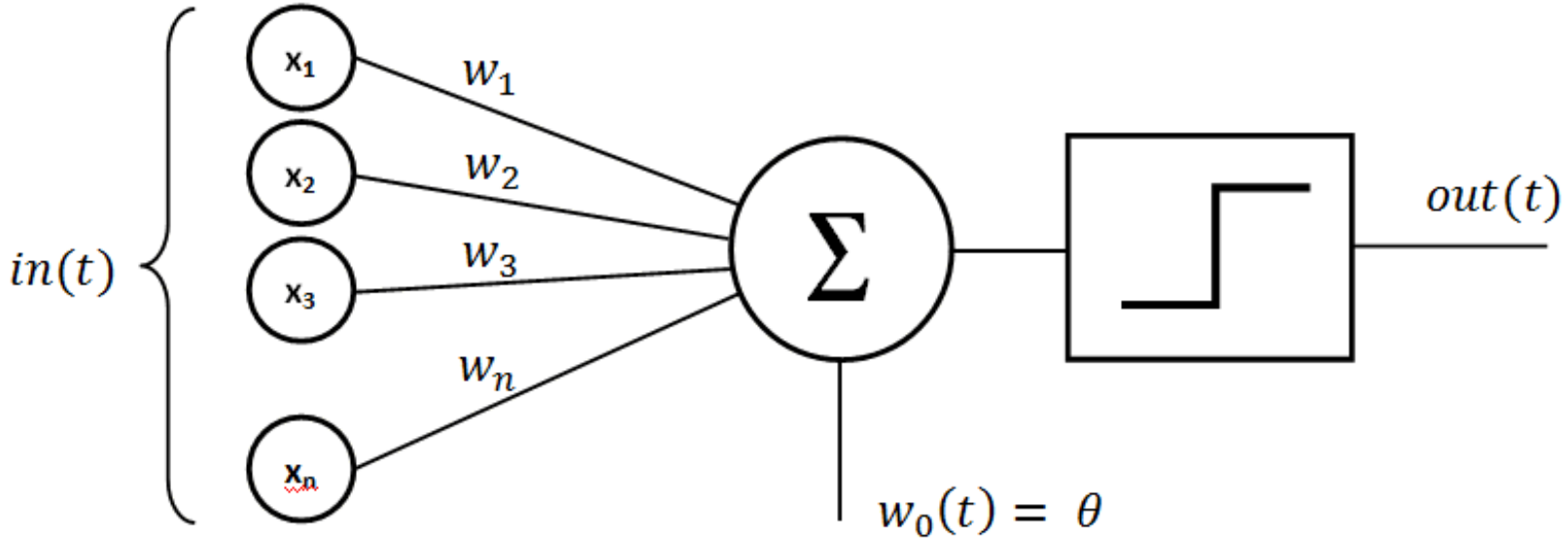


Desenvolvimento de redes neurais com Flux

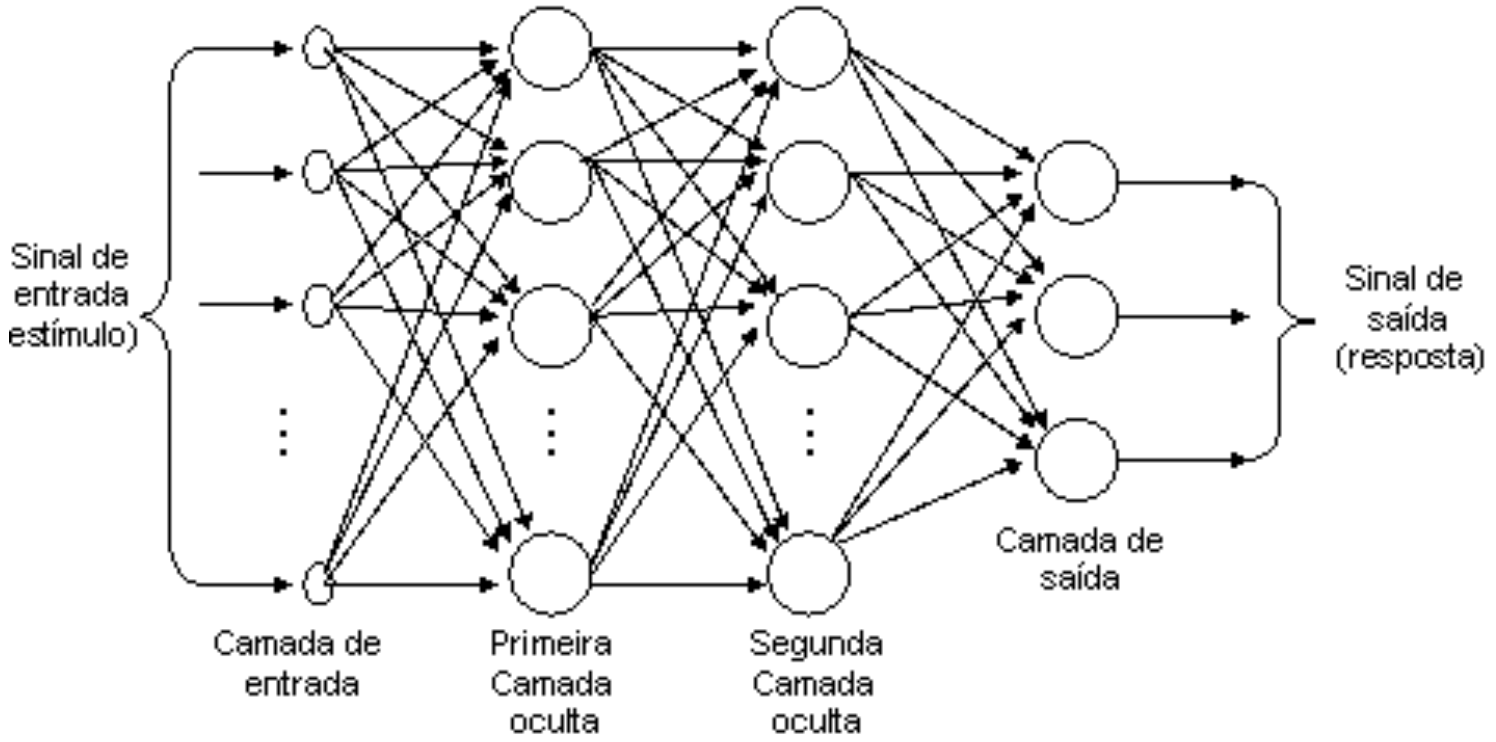
Desenvolvimento de uma rede neural simples para classificação de dados utilizando o conceito de perceptron.

O que é um perceptron?



$$y(X) = \varphi\left(W_0 + \sum_{i=1}^{n} W_i X_i\right)$$

Relação de múltiplas camadas



Treinamento supervisionado da rede

O treinamento supervisionado é aplicado corrigindo os pesos conforme o erro calculado por uma função de perda e uma função de optimização.

$$\Delta w_{j,i}(t) = \eta \delta_j(t) y_i(t)$$

η é o parâmetro da taxa de aprendizagem, $\delta_j(t)$ é o gradiente local no neurônio j e $y_i(t)$ é o sinal de entrada do neurônio j.

$$\delta_j(t) = erro_j(t) \varphi'_j(v_j(t))$$

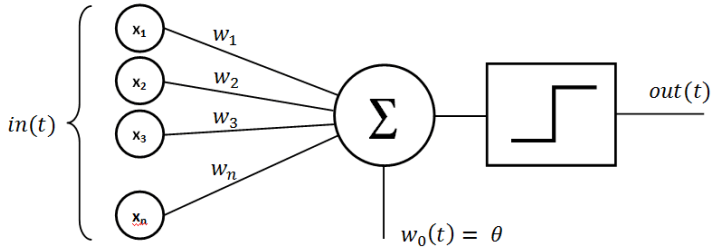
$$\delta_j(t) = \varphi'_j(v_j(t)) \sum_k \delta_k(t) W_{k,j}(t)$$

Conjunto de dados usado

n	ω	ϕ	ϕ
1	0.411	2.108	-1.258
2	5.566	3.051	0.612
3	1.624	0.438	2.511
4	0.213	2.812	-3.571
5	0.074	0.810	5.122
6	3.561	0.579	0.023
7	1.222	-2.830	1.113
8	2.667	-4.701	-0.782
9	0.316	0.865	2.367
10	0.251	0.357	3.157

```
1   using Flux, Plots, Statistics, ProgressMeter
2
3   ω_list = [0.411, 5.566, 1.624, 0.213, 0.074, 3.561, 1.222, 2.667, 0.316, 0
4   ϕ_list = [2.108, 3.051, 0.438, 2.812, 0.810, 0.579, -2.830, -4.701, 0.865,
5
6   times = range(0, stop=10, length=1000)
7
8   data = Matrix{Float32}(undef, 200, 1000)
9
10  for i = 1:10
11      for j = 1:20
12          for p = 1:1000
13              ω = ω_list[i]
14              ϕ = ϕ_list[j]
15              local t = times[p]
16              data[(i - 1) * 10 + j, x] = sin(ω * t + ϕ)
17          end
18      end
19  end
```

Criando o modelo com o Flux




```julia
function linear(in, out)
  W = randn(out, in)
  b = randn(out)
  x -> W * x .+ b
end

linear1 = linear(5, 3) # we can access linear1.W etc
linear2 = linear(3, 2)

model(x) = linear2(σ.(linear1(x)))

model(rand(5)) # => 2-element vector
```

Documentação do Flux - https://fluxml.ai/Flux.jl/stable/models/basics/

Criando o modelo com o Flux

```
layer1 = Dense(10 => 5, σ)
# ...
model(x) = layer3(layer2(layer1(x)))
```

```
using Flux

layers = [Dense(10 => 5, σ), Dense(5 => 2), softmax]

model(x) = foldl((x, m) -> m(x), layers, init = x)

model(rand(10)) # => 2-element vector
```

Documentação do Flux - https://fluxml.ai/Flux.jl/stable/models/basics/

Criando o modelo com o Flux


```
model2 = Chain(
  Dense(10 => 5, σ),
  Dense(5 => 2),
  softmax)

model2(rand(10)) # => 2-element vector
```

Documentação do Flux - https://fluxml.ai/Flux.jl/stable/models/basics/

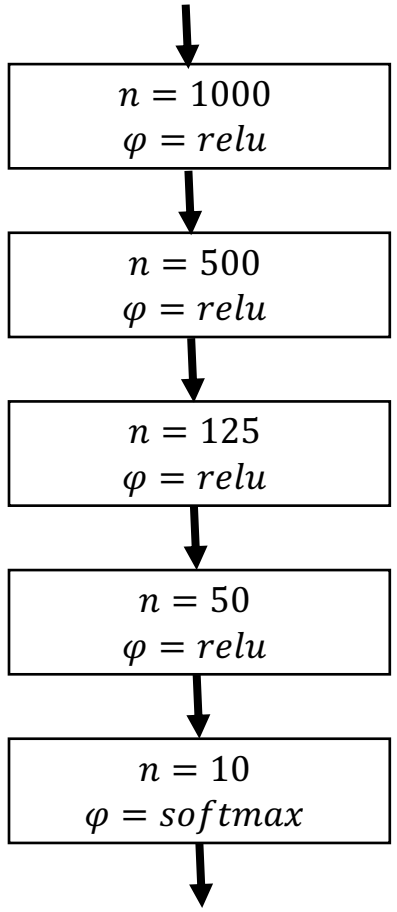
https://fluxml.ai/Flux.jl/stable/models/layers/

Criando o modelo com o Flux

```
105    model = Chain(
106        Dense(1000, 500, relu),
107        Dense(500, 125, relu),
108        Dense(125, 50, relu),
109        Dense(50, 10),
110        softmax
111    )
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Progress: 100%|
Chain(
  Dense(1000 => 500, relu),             # 500_500 parameters
  Dense(500 => 125, relu),              # 62_625 parameters
  Dense(125 => 50, relu),               # 6_300 parameters
  Dense(50 => 10),                      # 510 parameters
  NNlib.softmax,
)                   # Total: 8 arrays, 569_935 parameters, 2.175 MiB.
```



Tratando a saída da rede

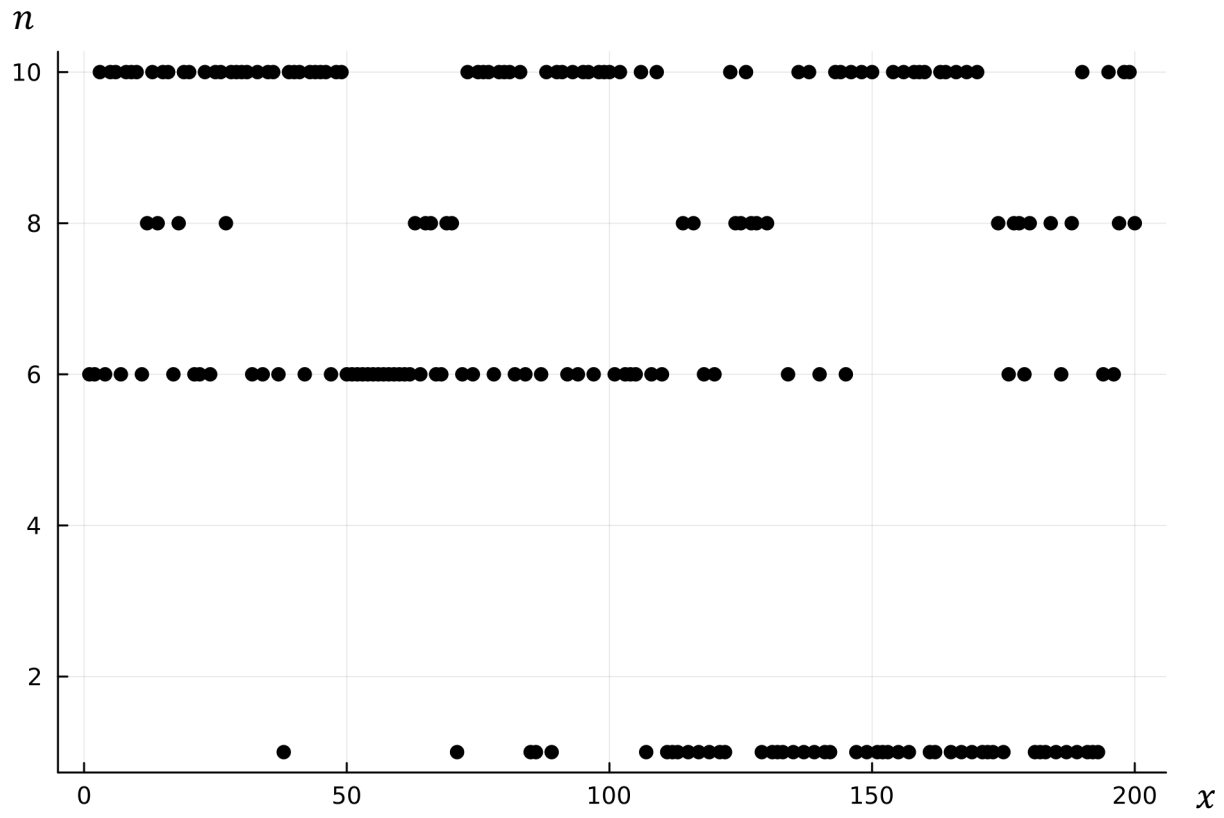
$$\phi(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{n} e^{z_k}}$$

```
10×200 Matrix{Float32}:
 0.101192   0.098345   0.158601   0.097925
 0.0878354  0.0826372  0.0773421  0.086412
 0.0878354  0.0826372  0.0773421  0.086412
 0.0878354  0.0826372  0.0836591  0.086412
 0.0878354  0.0826372  0.0773421  0.086412
 0.121717   0.160115   0.0773421  0.150008
 0.0878354  0.0826372  0.0773421  0.086412
 0.0878354  0.0826372  0.0950444  0.086412
 0.138178   0.0850244  0.0896039  0.0921529
 0.111901   0.160693   0.186381   0.141443
```

```
function get_graphdata_from_output(output)
    result = []
    for i = 1:200
        maior = 1
        if (output[2, i] > output[maior, i]) maior = 2 end
        if (output[3, i] > output[maior, i]) maior = 3 end
        if (output[4, i] > output[maior, i]) maior = 4 end
        if (output[5, i] > output[maior, i]) maior = 5 end
        if (output[6, i] > output[maior, i]) maior = 6 end
        if (output[7, i] > output[maior, i]) maior = 7 end
        if (output[8, i] > output[maior, i]) maior = 8 end
        if (output[9, i] > output[maior, i]) maior = 9 end
        if (output[10, i] > output[maior, i]) maior = 10 end
        push!(result, maior)
    end
    return result
end
```

Teste sem treinamento

n	ω
1	0.411
2	5.566
3	1.624
4	0.213
5	0.074
6	3.561
7	1.222
8	2.667
9	0.316
10	0.251



Treinamento da rede

1. Carregar a estrutura de dados com o Flux.

```
38    desired_values = get_desired_values()
39    loader = Flux.DataLoader((data', desired_values), batchsize = 64, shuffle = true)
```

2. Definir a função de optimização.

```
41    optim = Flux.setup(Adam(0.01), model)
```

https://fluxml.ai/Flux.jl/stable/training/optimisers/

3. Definir a função de perda.

```
42    loss(y_hat,y) = Flux.crossentropy(y_hat, y)
```

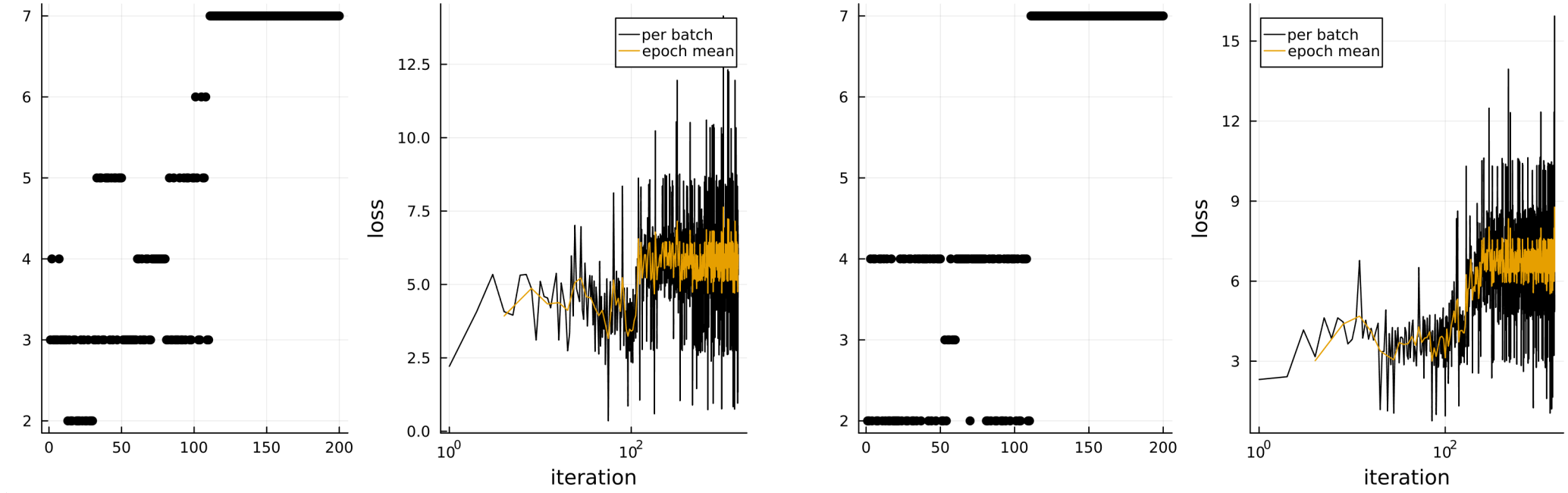
https://fluxml.ai/Flux.jl/stable/models/losses/

Treinamento da rede

4. Determina uma repetição de treinamento por um período.

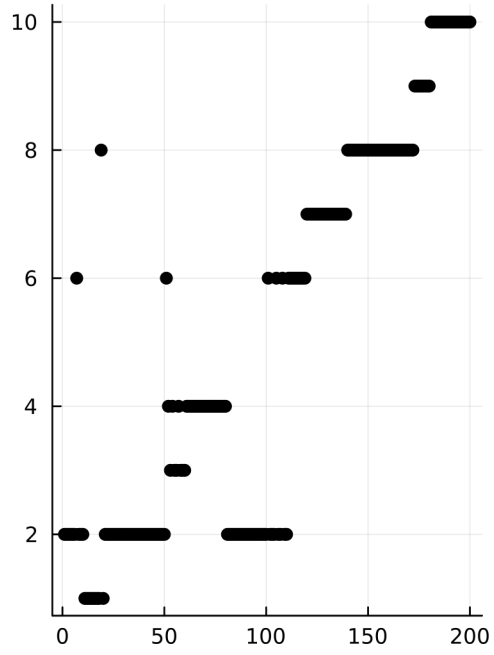
```
122  losses = []
123  @showprogress for epochs = 1:200
124      for (x, y) in loader
125          loss_r, grads = Flux.withgradient(model) do m
126              y_hat = m(x)
127              loss(y_hat, y)
128          end
129          Flux.update!(optim, model, grads[1])
130          push!(losses, loss_r)
131      end
132  end
```

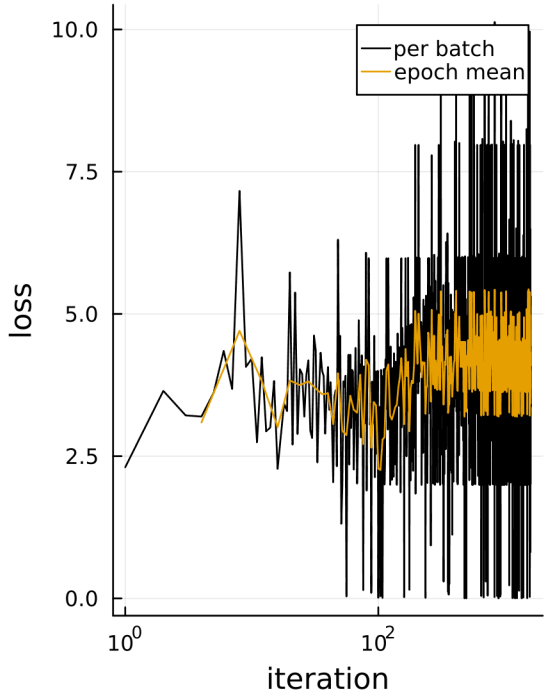
Testando a rede treinada

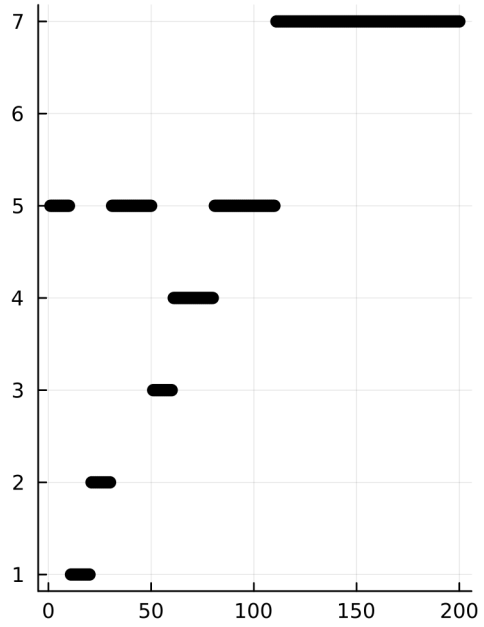


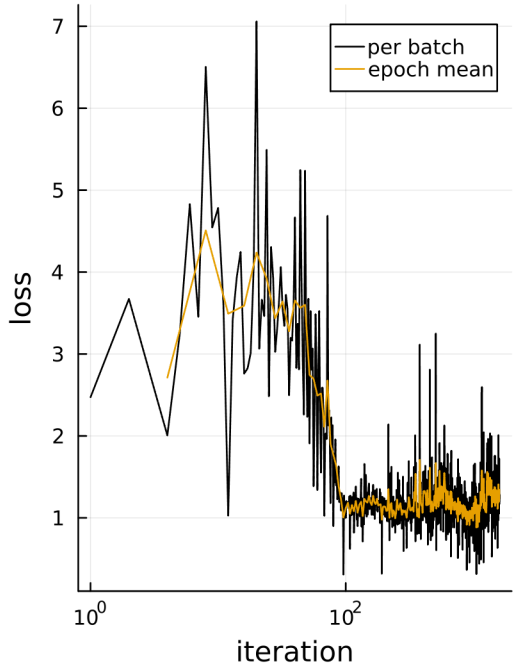
375 épocas

Testando a rede treinada



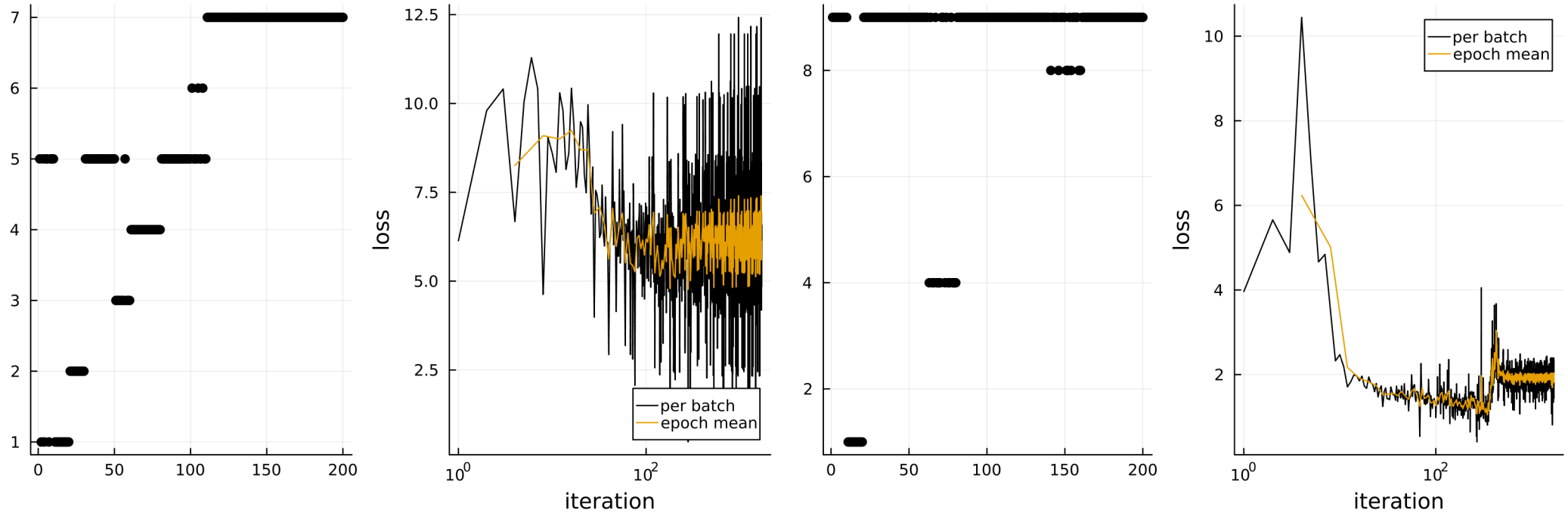






400 épocas

Testando a rede treinada



425 épocas

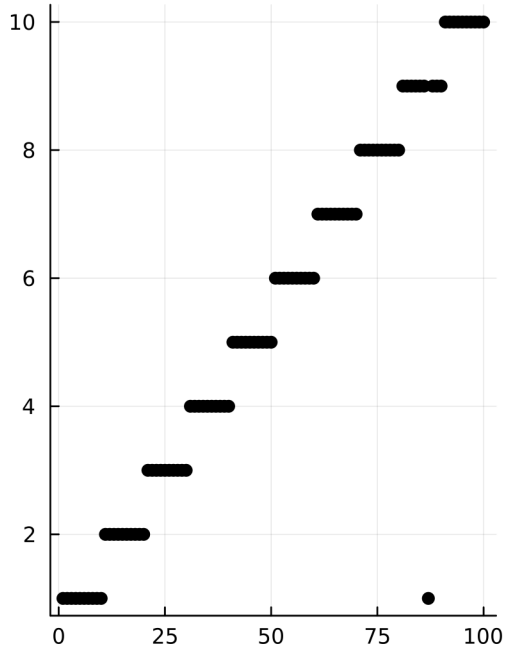
Testando a rede treinada

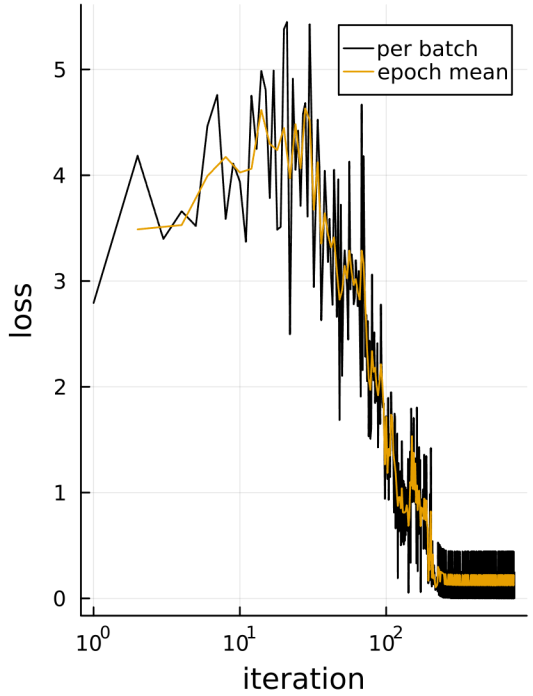
Gerados a partir de uma distribuição normal.

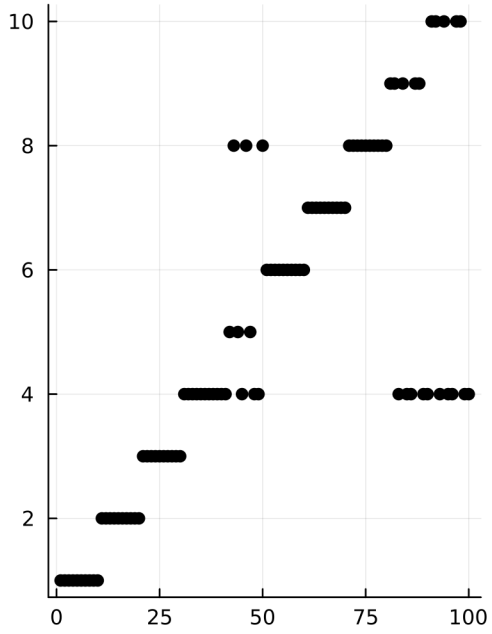
n	ω	ϕ	ϕ
1	0.411	2.108	-1.258
2	5.566	3.051	0.612
3	1.624	0.438	2.511
4	0.213	2.812	-3.571
5	0.074	0.810	5.122
6	3.561	0.579	0.023
7	1.222	-2.830	1.113
8	2.667	-4.701	-0.782
9	0.316	0.865	2.367
10	0.251	0.357	3.157

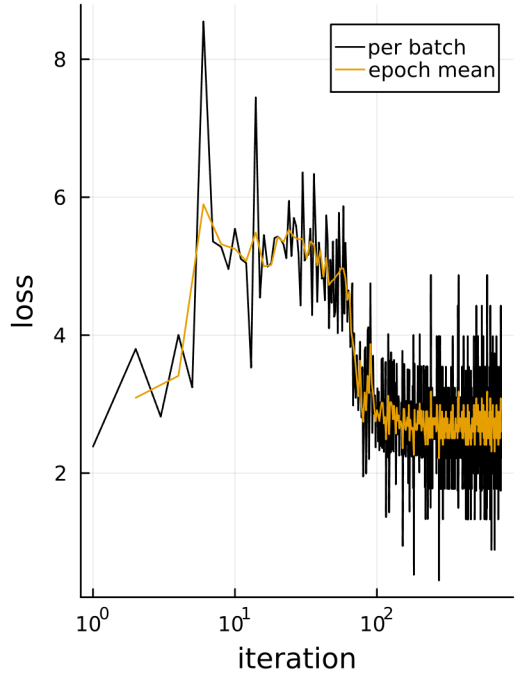
Gerados sem nenhuma relação.

Testando a rede treinada



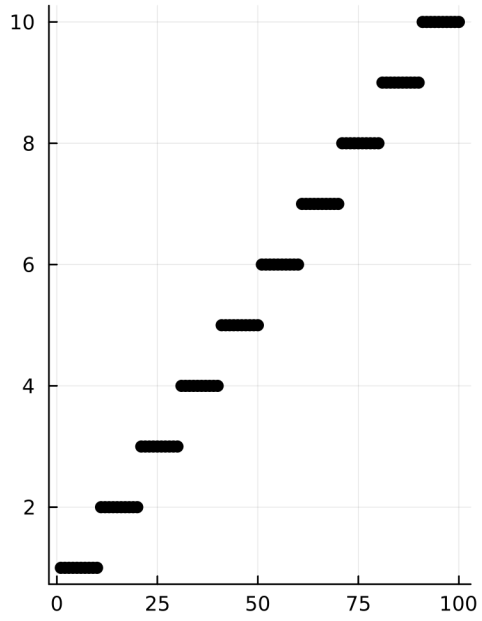


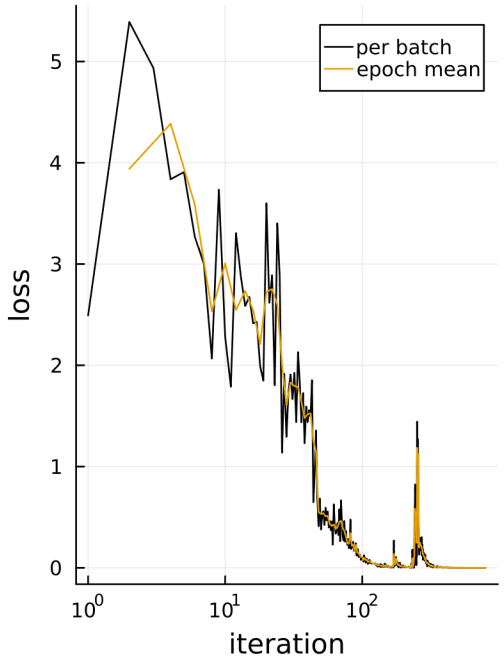


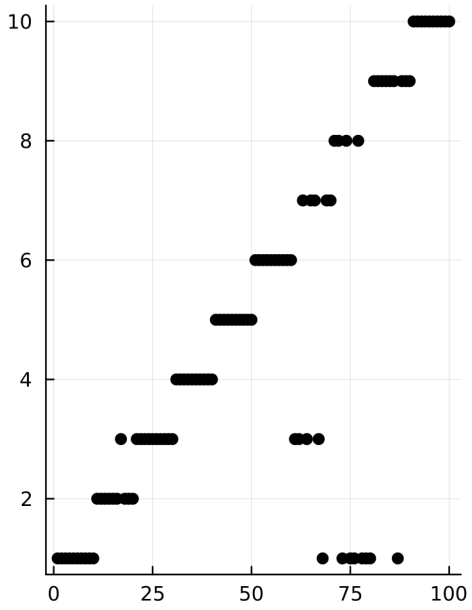


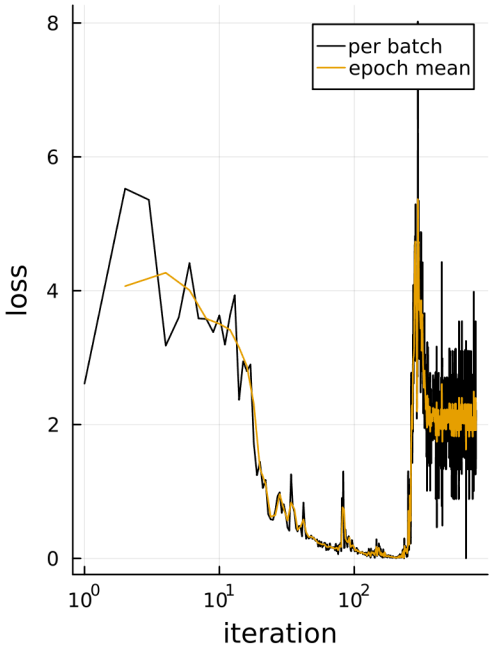
375 épocas

Testando a rede treinada



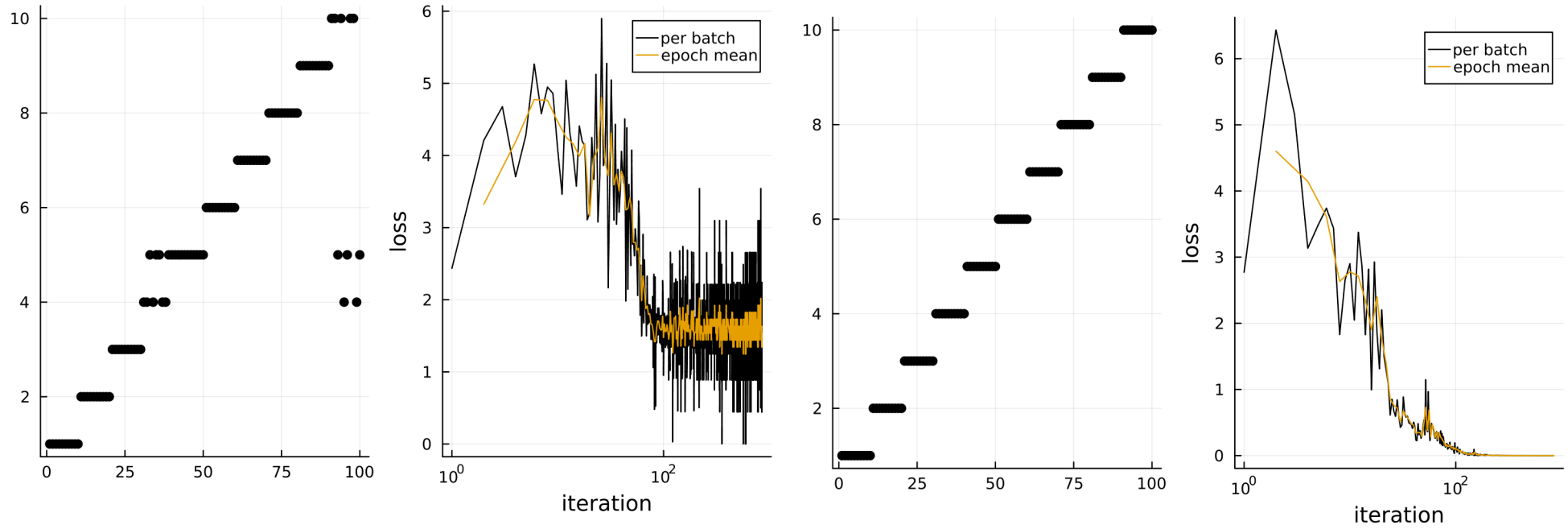






400 épocas

Testando a rede treinada



425 épocas

Utilizando a GPU

```
105    model = Chain(
106        Dense(1000, 500, relu),
107        Dense(500, 125, relu),
108        Dense(125, 50, relu),
109        Dense(50, 10),
110        softmax
111    ) |> gpu

117    loader = Flux.DataLoader((data', desired_values), batchsize = 64, shuffle = true) |> gpu

134    out_2 = get_graphdata_from_output(model(data'|>gpu)|>cpu)
135    graph_2 = scatter(x[1:100], out_2[1:100], legend=false)

 ┌ Info: The GPU function is being called but the GPU is not accessible.
 └ Defaulting back to the CPU. (No action is required if you want to run on the CPU).
Progress: 100%|

julia>
```

Referências

HAYKIN, Simon. Redes Neurais: princípios e prática, 2ª edição. Editora Bookman, 2001, reimpressão de 2008. ISBN 978-85-7307-718-6

Documentação, Flux, junho/2023. Disponível em: https://fluxml.ai/Flux.jl/stable/

Fim.
Muito obrigado!

https://github.com/gabriel-ferr/perceptron-seno