# MNIST classification using Neural Networks

```
TaskLocalRNG()
```

```
begin
    using Flux ✓
    using Plots ✓
    using Random ✓
    using Metrics ✓
    using MLUtils ✓
    using Statistics ✓
    using MLDatasets ✓
    using ProgressLogging ✓

    Random.seed!(9)
end
```

## Loading MNIST dataset

```
((28, 28, 60000), (60000))
```

```
begin
    train_x, train_y = MLDatasets.MNIST(:train)[:]
    size(train_x), size(train_y)
end
```

```
((28, 28, 10000), (10000))
```

```
begin
    test_x, test_y = MLDatasets.MNIST(:test)[:]
    size(test_x), size(test_y)
end
```

## Preparing data

```
(10×60000 OneHotMatrix(::Vector{UInt32}) with eltype Bool:
 ⋅  1  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  …  1  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅
 ⋅  ⋅  ⋅  1  ⋅  ⋅  1  ⋅  1  ⋅  ⋅  ⋅  ⋅  ⋅     ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  1  ⋅  ⋅  ⋅  ⋅  ⋅
 ⋅  ⋅  ⋅  ⋅  ⋅  1  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅     ⋅  ⋅  ⋅  ⋅  1  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅
 ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  1  ⋅  ⋅  1  ⋅  1  ⋅     ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  1  ⋅  ⋅  ⋅
 ⋅  ⋅  1  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  1  ⋅  ⋅  ⋅  ⋅     ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅
 1  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  1  ⋅  ⋅  …  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  1  ⋅  ⋅  ⋅  1  ⋅  ⋅
 ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  1     ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  1  ⋅
 ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅     ⋅  1  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅
 ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅     ⋅  ⋅  1  ⋅  ⋅  ⋅  ⋅  ⋅  1  ⋅  ⋅  ⋅  1
 ⋅  ⋅  ⋅  ⋅  1  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅     ⋅  ⋅  ⋅  1  ⋅  1  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅
```

```
ohb_train_y, ohb_test_y = Flux.onehotbatch(train_y, 0:9),
Flux.onehotbatch(test_y, 0:9)
```

```
train_data_loader =
60-element DataLoader(::Tuple{Array{Float32, 3}, OneHotArrays.OneHotMatrix{UInt32, Vec
  with first element:
  (28×28×1000 Array{Float32, 3}, 10×1000 OneHotMatrix(::Vector{UInt32}) with eltype Bo
```

```
train_data_loader = DataLoader((train_x, ohb_train_y), batchsize=1000)
```

# Defining model

```
Adam(0.001, (0.9, 0.999), 1.0e-8, IdDict())
```

```
begin
    model = Chain(
        Flux.flatten,
        Dense((train_x |> size)[1:2] |> prod => 64, relu),
        Dense(64 => 32, relu),
        Dense(32 => 10),
        softmax
    ) |> gpu
    optimizer = Adam()
end
```

```
Chain(
  Flux.flatten,
  Dense(784 => 64, relu),               # 50_240 parameters
  Dense(64 => 32, relu),                # 2_080 parameters
  Dense(32 => 10),                      # 330 parameters
  NNlib.softmax,
)                   # Total: 6 arrays, 52_650 parameters, 206.039 KiB.
```

```
model
```

```
loss (generic function with 1 method)
```

```
loss(x, y) = Flux.crossentropy(model(x), y) |> gpu
```

# Training phase

```
epochs = 20
```

```
epochs = 20
```

```
begin
    train_losses = Float32[]
    test_losses = Float32[]

    @progress for e in 1:epochs
        for d in train_data_loader
            gs = gradient(Flux.params(model)) do
                l = loss(d...)
            end
            Flux.update!(optimizer, Flux.params(model), gs)
        end
        push!(train_losses, loss(train_data_loader.data...))
        push!(test_losses, loss((test_x, ohb_test_y)...))
    end
end
```

100%

# Testing phase

```
pred =
```

```
10×10000 Matrix{Float32}:
 5.1544f-6    7.21666f-8   4.81226f-5   …  3.04162f-9   1.54637f-8   1.57561f-8
 7.46966f-8   2.32155f-5   0.983895        4.81534f-8   2.36768f-7   6.03511f-11
 1.52606f-5   0.999908     0.00100621      2.50781f-9   2.70912f-10  1.7688f-6
 0.00019571   4.78337f-5   0.000105507     2.63501f-8   1.10995f-6   1.05011f-9
 4.4628f-9    1.30821f-12  0.00255377      0.999779     7.98359f-9   5.48853f-6
 1.43595f-5   3.17883f-6   0.000708526  …  2.93158f-8   0.999881     3.18005f-7
 1.64995f-12  4.45508f-6   0.00152025      4.74957f-9   6.14238f-8   0.999992
 0.999707     9.89798f-12  0.0084575       1.2516f-6    5.03739f-8   1.6581f-12
 3.75128f-6   1.3633f-5    0.00163014      3.72587f-6   0.000117151  1.99344f-10
 5.86406f-5   2.80428f-10  7.52143f-5      0.000216185  1.6918f-8    6.11957f-11
```

```
pred = model(test_x)
```

```
true_pred =
```

```
▸ [7, 2, 1, 0, 4, 1, 4, 9, 6, 9, 0, 6, 9, 0, 1, 5, 9, 7, 3, 4, ⋯ more ,7, 8, 9, 0, 1, 2, 3
```

```
true_pred = Flux.onecold(pred, 0:9)
```

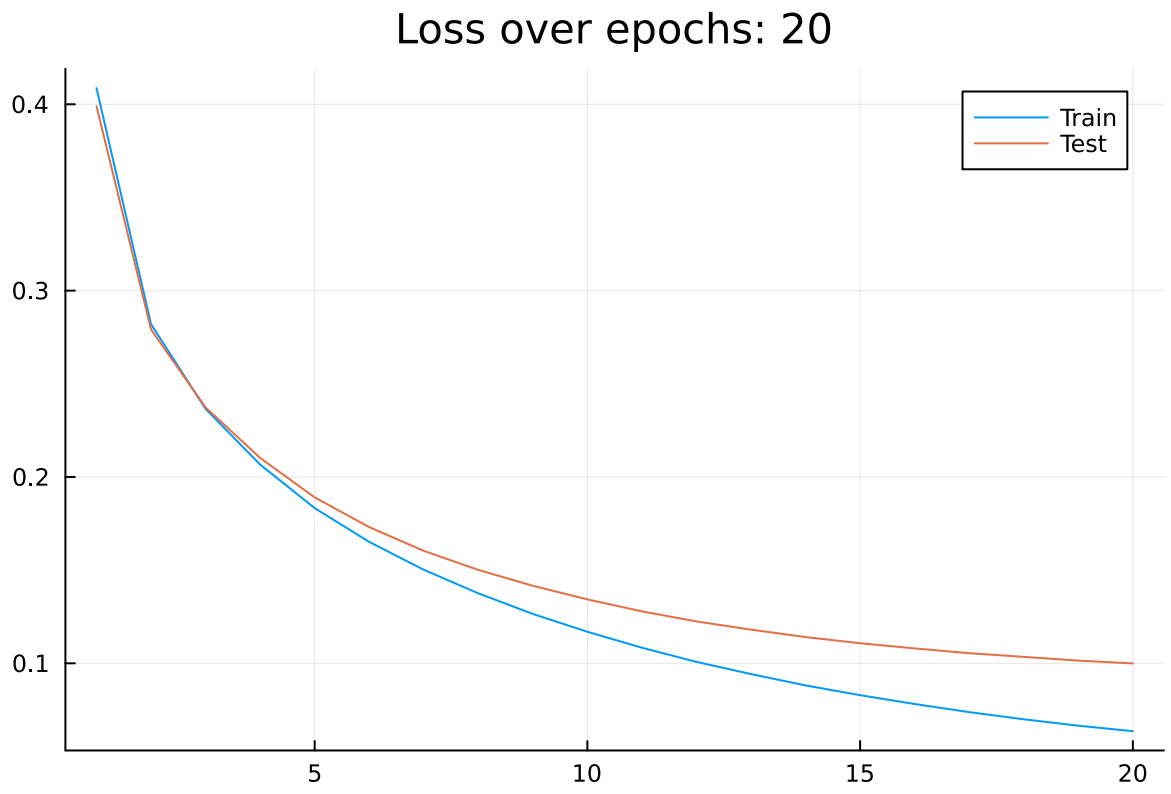
```
ohb_true_pred =
```

```
10×10000 OneHotMatrix(::Vector{UInt32}) with eltype Bool:
 ⋅  ⋅  ⋅  1  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  1  ⋅  ⋅  1  …  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  1  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅
 ⋅  ⋅  1  ⋅  ⋅  1  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅     ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  1  ⋅  ⋅  ⋅  ⋅  ⋅
 ⋅  1  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅     ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  1  ⋅  ⋅  ⋅  ⋅
 ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅     ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  1  ⋅  ⋅  ⋅
 ⋅  ⋅  ⋅  ⋅  1  ⋅  1  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅     1  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  1  ⋅  ⋅
 ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  …  ⋅  1  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  1  ⋅
 ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  1  ⋅  ⋅  1  ⋅  ⋅     ⋅  ⋅  1  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  1
 1  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅     ⋅  ⋅  ⋅  1  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅
 ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅     ⋅  ⋅  ⋅  ⋅  1  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅
 ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  1  ⋅  1  ⋅  ⋅  1  ⋅     ⋅  ⋅  ⋅  ⋅  ⋅  1  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅
```

```
ohb_true_pred = Flux.onehotbatch(true_pred, 0:9)
```

```
0.9694
```

```
Metrics.categorical_accuracy(ohb_true_pred, ohb_test_y)
```




```
begin
    plot(1:epochs, train_losses, title="Loss over epochs: $epochs", label="Train")
    plot!(1:epochs, test_losses, label="Test")
end
```