MNIST digits classification using a dnn with Flux.jl

Classifying MNIST digits dataset with a simple multi-layer-perceptron (MLP)

Table of Contents


```
using PlutoUI
```

```
PlutoUI.TableOfContents(indent=true, depth=4, aside=true)
```

```
# using Pkg; Pkg.activate("."); Pkg.instantiate()
```

```
begin
    using Flux    , Images    , MLDatasets    , Plots
    using Statistics
    using Random    , Statistics, LinearAlgebra
end
```

Load train and test data

```
((28, 28, 60000), (28, 28, 10000))
```

```
begin
    train_x, train_y = MNIST.traindata(Float32)
    test_x,  test_y  = MNIST.testdata(Float32)

    size(train_x), size(test_x)
end
```

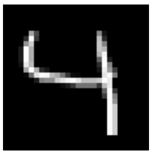
```
TaskLocalRNG()
```

```
Random.seed!(42)
```

Let's check an image, randomly picked in the train set.

```
ix = 6309
```

```
ix = Random.rand(1:size(train_x)[3])
```



```
begin
    img = train_x[:, :, ix]
    colorview(Gray, img')
end
```

Let's check the label for this image.

```
4
```

```
train_y[ix]
```

Prepare the data

As we are going to use a simple ANN for classifying the digits, we need to turn this gray images (of shape 28 x 28 x 1 - 1 being for the unique channel) into a vector of length 28 × 28 = 784.

To achieve this we need to flatten the images using Flux. The same processing needs to be applied for both the train and the test sets.

```
(
  1: 784×60000 Matrix{Float32}:
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0
      ⋮                        ⋮              ⋱                 ⋮
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0
  2: 784×10000 Matrix{Float32}:
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
      ⋮                        ⋮              ⋱                 ⋮
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
      0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
)
```

```julia
xtrain, xtest = Flux.flatten(train_x), Flux.flatten(test_x)
```

Table of Contents

For the labels we need to onehotencode them. This will be useful during the training stage to evaluate the loss function.

This needs to be apllied on both the train and test labels.

```
(10×60000 OneHotMatrix(::Vector{UInt32}) with eltype Bool:                        , 10×10000 OneHotM
 ⋅  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  ⋅  ⋅  ⋅  ⋅  ⋅  1  ⋅  ⋅  ⋅  1   ⋅  ⋅  ⋅  ⋅  ⋅
 ⋅  ⋅  ⋅  ⋅  1  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅     ⋅  ⋅  ⋅  1  ⋅  1  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅  ⋅   ⋅  ⋅  ⋅  ⋅  ⋅
```

```julia
ytrain, ytest = Flux.onehotbatch(train_y, 0:9), Flux.onehotbatch(test_y, 0:9)
```

Defining the model

```
const N_UNITS =  [48, 10]
```

```julia
const N_UNITS = [48, 10]  # 48 units in the 1st hidden layer and 10 in the output layer
```

```
(28, 28)
```

```julia
w, h = size(train_x)[1:2]
```

```
model = Chain(
          Dense(784, 48, relu),                 # 37_680 parameters
          Dense(48, 10),                        # 490 parameters
          NNlib.softmax,
        )                   # Total: 4 arrays, 38_170 parameters, 149.352 KiB.
```

```julia
model = Flux.Chain(
    Flux.Dense(w * h, N_UNITS[1], Flux.relu),
    Flux.Dense(N_UNITS...),  # no activation, ouptut the logit...
    Flux.softmax             # and finally  apply the softnax activation
)
```

Define the loss

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

```julia
loss(x, y) = Flux.Losses.crossentropy(model(x), y)
```

Get the parameters of the model

```julia
prms = Flux.params(model);  # The parameters of the model ≡ 2 weights matrices and 2 bias vectors
```

```
((48, 784), (-0.0849188, 0.0849124))
```

```julia
prms[1] |> size, extrema(prms[1])  # weight matrix for 1st layer
```

```
(48)
```

```julia
prms[2] |> size  # the bias vector for the 1st layer
```

```
((10, 48), (-0.319879, 0.317753))
```

```
prms[3] |> size, extrema(prms[3]) # the weigth matrix for the 2nd (output) layer
```

```
(10)
```

```
prms[4] |> size  # the bias vector for the 2nd (output and last) layer
```

Table of Contents

Define the optimizer

```
ADAM(0.01, (0.9, 0.999), 1.0e-8, IdDict())
```

```
begin
    η = 0.01
    opt = Flux.ADAM(η)
end
```

Training the model

```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,    m
```

```
begin
    const EPOCHS = 500
    loss_history = fill(zero(Float32), EPOCHS)
end
```

```
for epoch ∈ 1:EPOCHS
    Flux.train!(loss, prms, [(xtrain, ytrain)], opt)
    loss_history[epoch] = loss(xtrain, ytrain)
    if epoch % 50 == 0
        println("Epoch: $(epoch) training loss: $(loss_history[epoch])")
    end
end
```

```
Epoch: 50 training loss: 0.20242682
Epoch: 100 training loss: 0.12862355
Epoch: 150 training loss: 0.09023971
Epoch: 200 training loss: 0.06655575
Epoch: 250 training loss: 0.050638746
Epoch: 300 training loss: 0.039145
Epoch: 350 training loss: 0.03056464
Epoch: 400 training loss: 0.024015589
Epoch: 450 training loss: 0.018978296
Epoch: 500 training loss: 0.014968148
```

Let's determine the accuracy of this model

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

```
begin
    ŷ_raw = model(xtest)
    ŷ = Flux.onecold(ŷ_raw) .- 1
end
```

```
0.9657
```

```
begin
    y = Flux.onecold(ytest) .- 1
    Statistics.mean(ŷ .== y)
end
# 0.9608 with 32 units in hidden layer
```

```
check =
BitVector: [true, true, true, true, true, true, true, true, true, true, true, true, true, true, true, tr
```

```
# display some results
check = ŷ .== y
```

```
# TODO: check errors
```

Let's plot the loss (during the training)

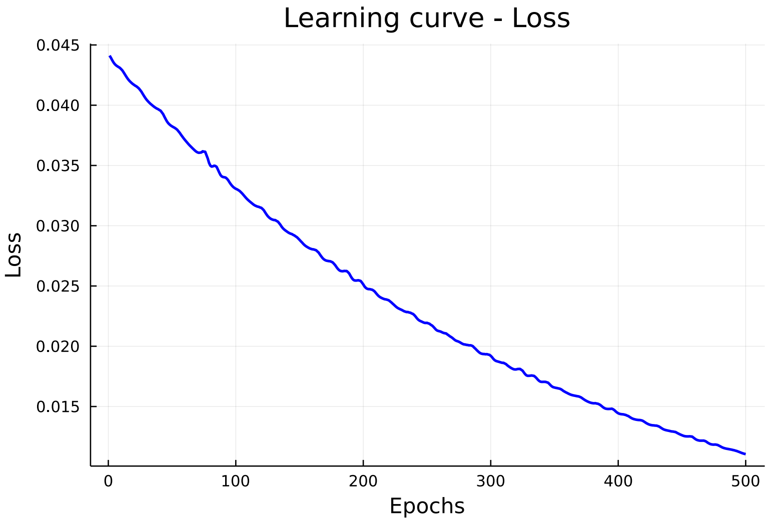


Table of Contents


```
begin
    gr(size = (600, 400))
    loss_curve = plot(
        1:EPOCHS,
        loss_history,
        xlabel = "Epochs",
        ylabel = "Loss",
        title = "Learning curve - Loss",
        legend = false,
        color = :blue,
        linewidth = 2
    )
end
```

```
html"""
<style>
  main {
    max-width: calc(800px + 25px + 6px);
  }

  .plutoui-toc.aside {
    background-color: linen;
    color: black;
  }

  h4, h5 {
    background: wheat;
    text-decoration: underline overline dotted darkred;
  }
</style>
"""
```