UPN Intelligent Systems Course Final Grade Project 2020

Premise

Train a NN to determine if student will approve a course, based on this rules:

- Consider three test results.
- The student will have almost two approved units to pass the course.
- To get the final result, there's no math calculation. Just take the mentioned previously.
- Results should be between 0 to 20.

```
TaskLocalRNG()
```

```
begin
    using Flux ✓
    using Plots ✓
    using Random ✓
    using Metrics ✓
    using MLUtils ✓
    using Statistics ✓
    using ProgressLogging ✓
    Random.seed!(9)
end
```

Generating grades data

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

```
generatedata(::Int64) = rand(0:20, 3) |> Vector{Int32}
```

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

```
checkcriteria(t) = (map(x -> x>11, t) |> sum > 1) |> Int32
```

```
n_student = 100000
```

```
n_student = 100_000
```

```
generated_data =
```

```
[[4, 4, 13], [9, 10, 10], [6, 2, 14], [8, 7, 7], [12, 15, 9], [16, 2, 7], [6, 5, 17], [17, 1
```

```
generated_data = map(generatedata, 1:n_student)
```

```
data_x =
3×100000 Matrix{Int32}:
  4   9   6  8  12  16   6  17  12  15  …  4  5   6  20   8   1  20   3  19  18  7
  4  10   2  7  15   2   5  15   7  12     3  2   2   0   3  10   9   2  16   4  0
 13  10  14  7   9   7  17   8  11  15     7  5  17  12  17  17   4  16   4   8  5
```

```
data_x = reduce(hcat, generated_data)
```

```
data_y =
1×100000 Matrix{Int32}:
 0  0  0  0  1  0  0  1  0  1  0  1  0  1  …  0  1  0  0  0  1  0  0  0  0  1  0  0
```

```
data_y = generated_data' .|> checkcriteria
```

Preparing data

```
(3×100000 view(::Matrix{Int32}, :, [36592, 35474, 60044, 77243, 47158, 80707, 14243, 4103
  12   4  3  11  12  17  16  18  10  3  14  …  19  19   8   5  18  12  16  19  18  11
   7   8  0  18  13  19   6  17  17  4  10      2  11  11   5  12  13   4  10   4   8
  17  16  5   2   3  17   5  12   9  9   6      3  13   1  20  16  12   8   5   0   6
```

```
s_data_x, s_data_y = MLUtils.shuffleobs((data_x, data_y))
```

```
((3×80000 view(::Matrix{Int32}, :, [36592, 35474, 60044, 77243, 47158, 80707, 14243, 4103
  12   4  3  11  12  17  16  18  10  3  14  …  15   7  7  11   2  20  12  16   6   5
   7   8  0  18  13  19   6  17  17  4  10     10  20  9  14  16   2  10  10   1   9
  17  16  5   2   3  17   5  12   9  9   6     14  15  0  11   6  19  19  11  17  20
```

```
train_data, test_data = MLUtils.splitobs((s_data_x, s_data_y); at=0.8)
```

```
train_data_loader =
80-element DataLoader(::Tuple{SubArray{Int32, 2, Matrix{Int32}, Tuple{Base.Slice{Base.OneT
  with first element:
  (3×1000 Matrix{Int32}, 1×1000 Matrix{Int32},)
```

```
train_data_loader = DataLoader(train_data, batchsize=1000)
```

```
test_data_loader =
20000-element DataLoader(::Tuple{SubArray{Int32, 2, Matrix{Int32}, Tuple{Base.Slice{Base.O
  with first element:
  (3×1 Matrix{Int32}, 1×1 Matrix{Int32},)
```

```
test_data_loader = DataLoader(test_data)
```

Defining model

```
model = Chain(
          Dense(3 => 16, relu),                 # 64 parameters
          Dense(16 => 8, relu),                 # 136 parameters
          Dense(8 => 1, σ),                     # 9 parameters
        )                   # Total: 6 arrays, 209 parameters, 1.191 KiB.
```

```
model = Chain(
    Dense(3 => 16, relu),
    Dense(16 => 8, relu),
    Dense(8 => 1, sigmoid)
) |> gpu
```

```
optimizer = Adam(0.05, (0.9, 0.999), 1.0e-8, IdDict())
```

```
optimizer = Adam(0.05)
```

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

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

Training phase

```
epochs = 100
```

```
epochs = 100
```

```
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_data_loader.data...))
    end
end
```

```
100%
```

Testing phase

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

```
accuracy(y, Y) = mean(Y .== y)
```

```
begin
    test_pred = Int32[]
    test_truth = test_data_loader.data[2]

    for (x, y) in test_data_loader
        pred = round(Int32, model(x)[1])
        append!(test_pred, pred)
    end
end
```

```
incorrect_predictions = 6
```

```
incorrect_predictions = test_truth .- test_pred' |> sum |> abs
```

```
@info "Correct predictions: $(size(test_pred)[1] - incorrect_predictions) -
Incorrect: $incorrect_predictions"
```

```
Correct predictions: 19994 - Incorrect: 6
```

```
mae = 0.0003
```

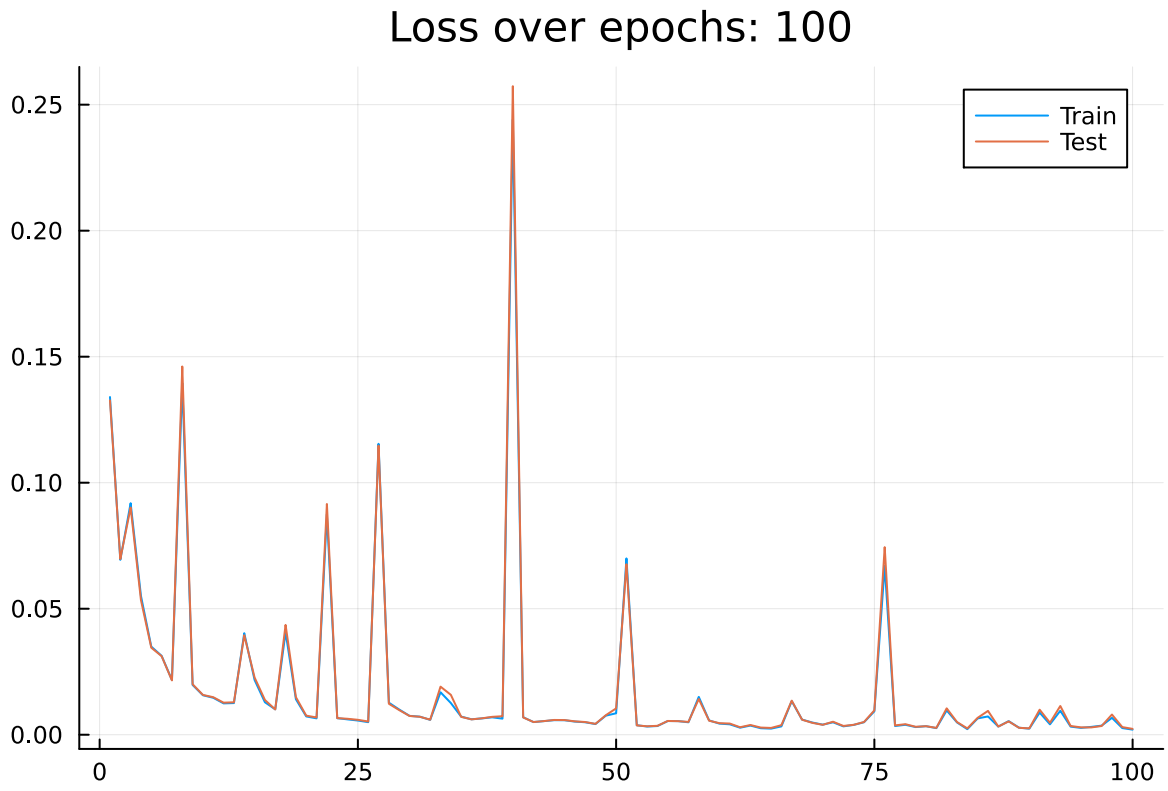
```
mae = Metrics.mae(test_pred', test_truth)
```

```
mse = 0.0003
```

```
mse = Metrics.mse(test_pred', test_truth)
```

```
acc = 0.9997
```

```
acc = accuracy(test_pred', test_truth)
```




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