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()
```

```julia
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)
```

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

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

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

```
n_student = 100000
```

```julia
n_student = 100_000
```

```
generated_data =
▸ [[6, 4, 11], [0, 1, 19], [20, 13, 11], [7, 19, 3], [5, 7, 18], [5, 7, 19], [12, 5, 8], [
```

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

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

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

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

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

Preparing data

```
norm_data_x =
3×100000 Matrix{Float32}:
 -0.659397  -1.64905  1.64979   -0.494455  …  -1.48411     -1.31917   -0.164571
 -0.984722  -1.48022  0.50176    1.49275       0.00626635  -1.48022    0.83209
  0.168894   1.4907   0.168894  -1.15291      -1.48336      0.995021   1.32547
```

```
norm_data_x = Flux.normalise(data_x) |> Matrix{Float32}
```

```
(3×100000 view(::Matrix{Float32}, :, [46711, 74525, 75829, 63236, 81724, 2524, 91360
   -0.494455     0.330255  -0.659397  -1.48411  …  -0.659397  -0.659397  -1.48411
   -0.654392    -0.489228  -0.489228  -1.14989     -0.158898   1.49275   -0.489228
    0.00366801  -0.657234  -1.64859   -1.31814      0.334119  -0.492009   1.65592
```

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

```
((3×80000 view(::Matrix{Float32}, :, [46711, 74525, 75829, 63236, 81724, 2524, 91360
   -0.494455     0.330255  -0.659397  -1.48411  …  -0.989282  0.825082   1.15497
   -0.654392    -0.489228  -0.489228  -1.14989     -0.158898  1.16242   -1.14989
    0.00366801  -0.657234  -1.64859   -1.31814      0.499345  1.65592    0.499345
```

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

```
train_data_loader =
80-element DataLoader(::Tuple{SubArray{Float32, 2, Matrix{Float32}, Tuple{Base.Slice{B
  with first element:
  (3×1000 Matrix{Float32}, 1×1000 Matrix{Int32},)
```

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

```
test_data_loader =
20000-element DataLoader(::Tuple{SubArray{Float32, 2, Matrix{Float32}, Tuple{Base.Slic
  with first element:
  (3×1 Matrix{Float32}, 1×1 Matrix{Int32},)
```

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

Defining model

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

```julia
begin
    model = Chain(
        Dense(3 => 8, relu),
        Dense(8 => 1, sigmoid)
    ) |> gpu
    optimizer = Adam(0.05)
end
```

```
Chain(
  Dense(3 => 8, relu),                  # 32 parameters
  Dense(8 => 1, σ),                     # 9 parameters
)                   # Total: 4 arrays, 41 parameters, 420 bytes.
```

```julia
model
```

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

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

Training phase

```
epochs = 15
```

```julia
epochs = 15
```

```julia
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)
```

```julia
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 = 0
```

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

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

```
Correct predictions: 20000 - Incorrect: 0
```

```
mae = 0.0
```

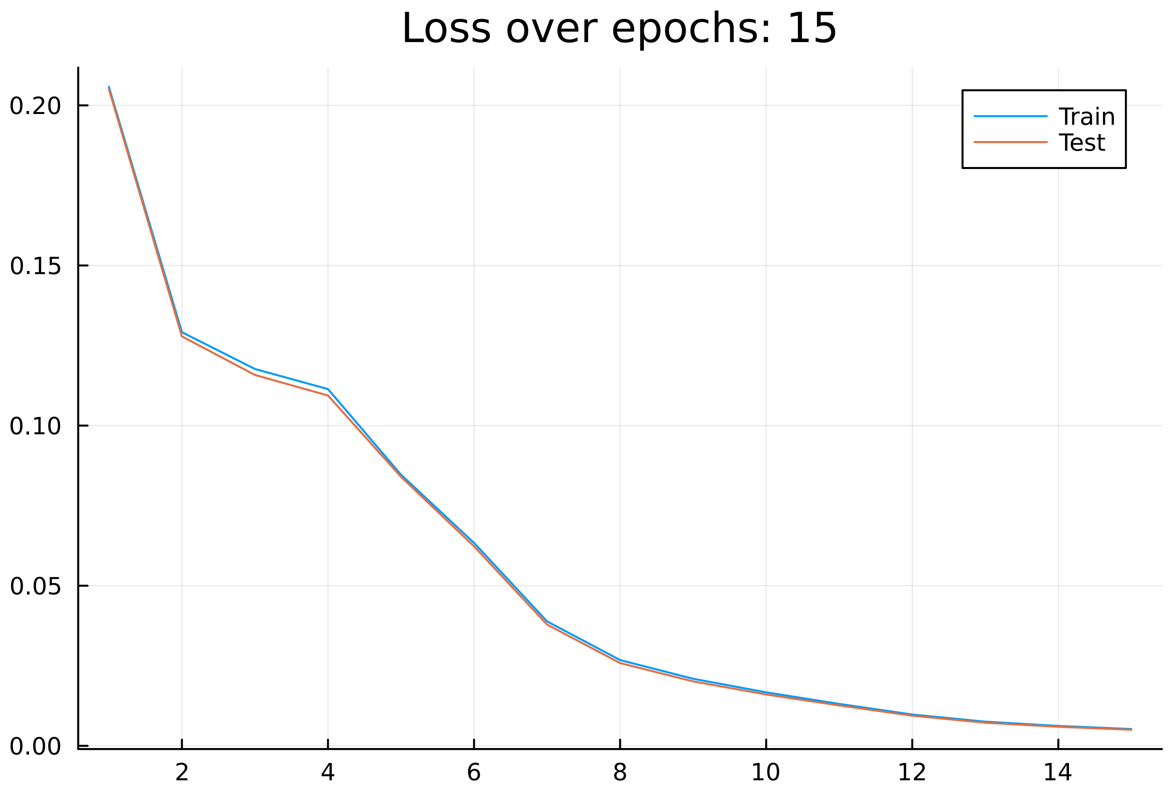
```
mae = Metrics.mae(test_pred', test_truth)
```

```
mse = 0.0
```

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

```
acc = 1.0
```

```
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
```