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 =
```

```
[[15, 4, 16], [10, 14, 7], [18, 7, 10], [10, 3, 12], [16, 4, 20], [20, 8, 3], [12, 0, 9]
```

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

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

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

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

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

Preparing data

```
norm_data_x =
3×100000 Matrix{Float32}:
  0.827859   0.00169199   1.32356      0.00169199  …  -0.824475   0.166925   0.662625
 -0.988051   0.664125    -0.492398    -1.15327        -1.15327    0.829343  -0.657616
  0.988098  -0.496873    -0.00188261   0.328111       -0.826867  -1.65185   -0.166879
```

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

```
((3×80000 view(::Matrix{Float32}, :, 1:80000) with eltype Float32:          , 1×80000
     0.827859   0.00169199   1.32356     …  -0.824475    -0.163541  0.827859   1  0  0
    -0.988051   0.664125    -0.492398         0.00325479  -1.15327   1.65543
     0.988098  -0.496873    -0.00188261       1.31809     -1.48685   1.15309
```

```
train_data, test_data = MLUtils.splitobs((norm_data_x, 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())
```

```
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.
```

```
model
```

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

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

Training phase

```
epochs = 15
```

```
epochs = 15
```

```
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 = 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
```

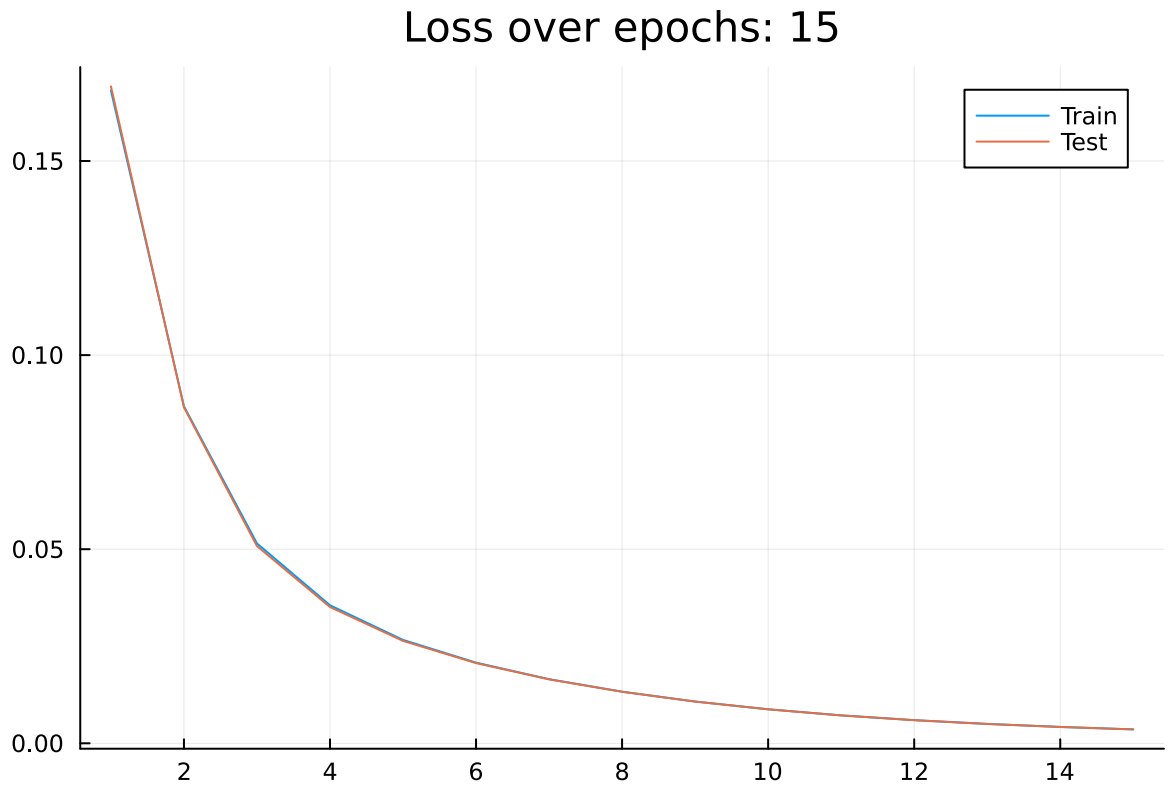
```
mae = Metrics.mae(test_pred', test_truth)
```

```
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
```