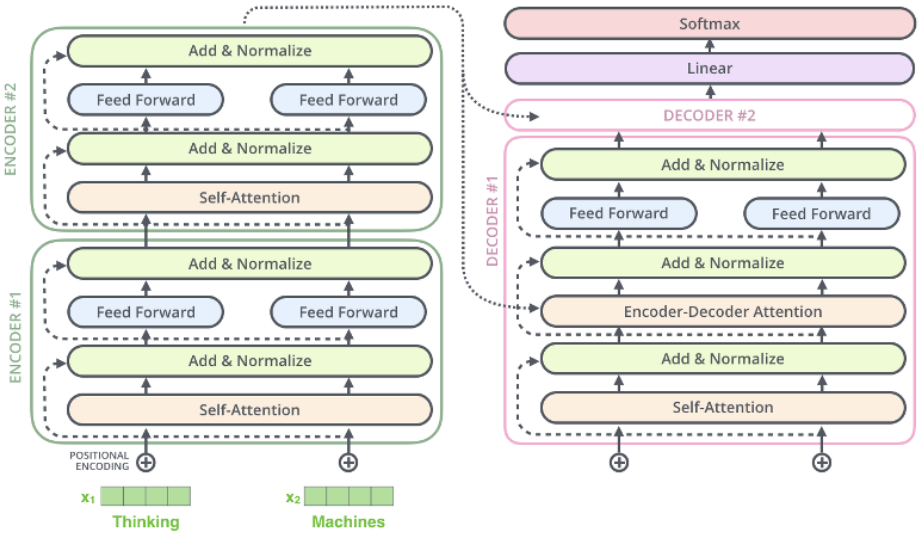


Sentiment analysis using a transformer

Model evaluation

Table of Contents

Restart

```
begin
    import Pkg
    Pkg.activate(mktempdir())
    Pkg.develop(url="https://github.com/rgreilly/Transformers")
    Pkg.add(["Revise", "PlutoUI", "Flux", "Plots", "DataFrames", "Printf",
        "BSON", "JSON", "Arrow", "StatsBase", "Unicode", "Random",
        "DataStructures", "ProgressMeter", "RemoteFiles"])

    using Revise
    using TransformersLite
    using PlutoUI
    using Flux
    using Flux: DataLoader
    using Plots
    using DataFrames
    using Printf
    using BSON, JSON
    using Arrow
    using StatsBase
    using Unicode
    using Random
    using DataStructures
    using ProgressMeter
    using RemoteFiles
    using TokenizersLite
end;
```

```
  Activating new project at `/tmp/jl_okGJon`
     Cloning git-repo `https://github.com/rgreilly/Transformers`
Path `/home/thomas/.julia/dev/TransformersLite` exists and looks like the cor
rect repo. Using existing path.
   Resolving package versions...
    Updating `/tmp/jl_okGJon/Project.toml`
  [6579f8b0] + TransformersLite v0.1.0 `~/.julia/dev/TransformersLite`
    Updating `/tmp/jl_okGJon/Manifest.toml`
  [621f4979] + AbstractFFTs v1.5.0
⌅ [79e6a3ab] + Adapt v3.7.2
  [dce04be8] + ArgCheck v2.3.0
  [69666777] + Arrow v2.7.0
  [31f734f8] + ArrowTypes v2.3.0
  [a9b6321e] + Atomix v0.1.0
  [ab4f0b2a] + BFloat16s v0.4.2
  [fbb218c0] + BSON v0.3.7
  [198e06fe] + BangBang v0.3.39
  [9718e550] + Baselet v0.1.1
  [c3b6d118] + BitIntegers v0.3.1
⌅ [fa961155] + CEnum v0.4.2
⌅ [052768ef] + CUDA v4.4.1
  [1af6417a] + CUDA_Runtime_Discovery v0.2.2
  [082447d4] + ChainRules v1.58.1
  [d360d2e6] + ChainRulesCore v1.19.0
  [5ba52731] + CodecLz4 v0.4.1
  [6b39b394] + CodecZstd v0.8.1
  [bbf7d656] + CommonSubexpressions v0.3.0
  [34da2185] + Compat v4.10.1
  [a33af91c] + CompositionsBase v0.1.2
  [f0e56b4a] + ConcurrentUtilities v2.3.0
  [187b0558] + ConstructionBase v1.5.4
  [6add18c4] + ContextVariablesX v0.1.3
  [a8cc5b0e] + Crayons v4.1.1
  [9a962f9c] + DataAPI v1.15.0
  [a93c6f00] + DataFrames v1.6.1
  [864edb3b] + DataStructures v0.18.15
  [e2d170a0] + DataValueInterfaces v1.0.0
  [244e2a9f] + DefineSingletons v0.1.2
  [8bb1440f] + DelimitedFiles v1.9.1
  [163ba53b] + DiffResults v1.1.0
  [b552c78f] + DiffRules v1.15.1
  [ffbed154] + DocStringExtensions v0.9.3
  [4e289a0a] + EnumX v1.0.4
  [e2ba6199] + ExprTools v0.1.10
  [cc61a311] + FLoops v0.2.1
  [b9860ae5] + FLoopsBase v0.1.1
  [1a297f60] + FillArrays v1.9.3
^ [587475ba] + Flux v0.13.17
  [f6369f11] + ForwardDiff v0.10.36
```

```
begin
    @RemoteFileSet FILES "Transformer utilities" begin
        reporting = @RemoteFile
    "https://github.com/rgreilly/Transformers/blob/main/examples/reporting.jl"
    dir="utilities" file="reporting.jl.json"
        utilities = @RemoteFile
    "https://github.com/rgreilly/Transformers/blob/main/examples/utilities.jl"
    dir="utilities" file="utilities.jl.json"
        training = @RemoteFile
        "https://github.com/rgreilly/Transformers/blob/main/examples/training.jl"
        dir="utilities" file="training.jl.json"
    end

    download(FILES) # Files downloaded in JSON format
end
```

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

```
function convertJSON(inFile, outFile)
    body = JSON.parsefile(inFile)["payload"]["blob"]["rawLines"]
    open(outFile, "w") do f
      for i in body
        println(f, i)
      end
    end
end
```

```
begin
    convertJSON("utilities/reporting.jl.json", "utilities/reporting.jl")
    convertJSON("utilities/utilities.jl.json", "utilities/utilities.jl")
    convertJSON("utilities/training.jl.json", "utilities/training.jl")

    include("utilities/reporting.jl")
    include("utilities/utilities.jl")
    include("utilities/training.jl")
end;
```

Multi-class classification: stars from 1 to 5

Load data

The original CSV data format has been converted to arrow format for faster loading.

```
begin
    path = "datasets/amazon_reviews_multi/en/1.0.0/"
    file_train = "train.arrow"
    file_test = "test.arrow"
    nlabels = 5
end;
```

Load training and test data into dataframes and about the size of both.

```
(205000, 5000)
```

```
begin
    filepath = joinpath(path, file_train)
    df = DataFrame(Arrow.Table(filepath))

    filepath = joinpath(path, file_test)
    df_test = DataFrame(Arrow.Table(filepath))

    (nrow(df), nrow(df_test))
end
```

Extract just the review text and the star rating

```
begin
    documents = df[:, "review_body"]
    labels = df[:, "stars"]

    println("training samples: ", size(documents), " ", size(labels))
end
```

```
training samples: (205000,) (205000,)
```

Do the same for the test data

```
begin
    documents_test = df_test[:, "review_body"]
    labels_test = df_test[:, "stars"];

    println("test samples: ", size(documents_test), " ", size(labels_test))
end
```

```
test samples: (5000,) (5000,)
```

Load the already trained and saved model. Note that models and associated details are stored in the `outputs` directory under a sub-directory name generated from the time it was saved in the format: `yyyymmdd_hhmm`.

```
begin
print(pwd(), "\n\n\n\n")
print(readdir(pwd()), "\n\n\n\n")
print(readdir("outputs"), "\n\n\n\n")
    print(readdir("outputs/20231228_1145/"))
end
```

```
/home/thomas/.julia/pluto_notebooks



["Cute journal.jl", "Cute program.jl", "Exciting creation.jl", "Fascinating exp
eriment.jl", "Fascinating program.jl", "Friendly analysis.jl", "Friendly experi
ment.jl", "Friendly report.jl", "Friendly revelation.jl", "Groundbreaking proo
f.jl"  …  "Tiny invention.jl", "Tiny revelation 1.jl", "Tiny revelation.jl", "W
ild blueprint.jl", "Wonderful analysis.jl", "Wonderful lecture.jl", "datasets",
"outputs", "utilities", "vocab"]



["20231221_1147", "20231228_1145", "20231228_1433", "20231231_1706"]



["Screenshot from 2023-12-28 14-00-28.png", "confusion_matrix.png", "history.js
on", "history.png", "hyperparameters.json", "model.bson", "prediction_star.pn
g"]
```

```
begin
    directory = "outputs/20231228_1145/"
    saved_objects = BSON.load(joinpath(directory, "model.bson"))
    tokenizer = saved_objects[:tokenizer]
    @show tokenizer
    indexer = saved_objects[:indexer]
    @show indexer
    model = saved_objects[:model]
    display(model)
end;
```

```
tokenizer = identity
indexer = IndexTokenizer{String}(length(vocabulary)=6654, unksym=[UNK])
TransformerClassifier(
  Embed((64, 6654)),                    # 425_856 parameters
  PositionEncoding(64),
  Dropout(0.25),
  TransformerEncoderBlock(
    MultiheadAttention(num_heads=8, head_size=8, 64=>64)(
      denseQ = Dense(64 => 64),         # 4_160 parameters
      denseK = Dense(64 => 64),         # 4_160 parameters
      denseV = Dense(64 => 64),         # 4_160 parameters
      denseO = Dense(64 => 64),         # 4_160 parameters
    ),
    Dropout(0.25),
    LayerNorm(64),                      # 128 parameters
    Dense(64 => 256, relu),             # 16_640 parameters
    Dense(256 => 64),                   # 16_448 parameters
    Dropout(0.25),
    LayerNorm(64),                      # 128 parameters
  ),
  TransformerEncoderBlock(
    MultiheadAttention(num_heads=8, head_size=8, 64=>64)(
      denseQ = Dense(64 => 64),         # 4_160 parameters
      denseK = Dense(64 => 64),         # 4_160 parameters
      denseV = Dense(64 => 64),         # 4_160 parameters
      denseO = Dense(64 => 64),         # 4_160 parameters
    ),
    Dropout(0.25),
    LayerNorm(64),                      # 128 parameters
    Dense(64 => 256, relu),             # 16_640 parameters
    Dense(256 => 64),                   # 16_448 parameters
    Dropout(0.25),
    LayerNorm(64),                      # 128 parameters
  ),
  Dense(64 => 1),                       # 65 parameters
  FlattenLayer(),
  Dense(50 => 5),                       # 255 parameters
)        # Total: 37 trainable arrays, 526_144 parameters,
          # plus 1 non-trainable, 64_000 parameters, summarysize 2.254 MiB.
```

Tokenise

Tokenise the training and test data

```
begin
    max_length = size(model.classifier.weight, 2)
    @time tokens = map(d->preprocess(d, tokenizer, max_length=max_length),
    documents) #takes about 30 seconds for all documents
    @time indices = indexer(tokens) #takes about 12 seconds for all documents

    y_train = copy(labels)
    idxs = Base.OneTo(length(labels))
    X_train, y_train = indices[:, idxs], y_train[idxs];
    y_train = Flux.onehotbatch(y_train, 1:5) # multi-class
    train_data, val_data = split_validation(X_train, y_train;
        rng=MersenneTwister(2718))

    println("train samples:      ", size(train_data[1]), " ", size(train_data[2]))
    println("validation samples: ", size(val_data[1]), " ", size(val_data[2]))
end
```

```
  5.179180 seconds (28.48 M allocations: 1.786 GiB, 16.40% gc time)
 22.166384 seconds (4 allocations: 79.765 MiB, 0.25% gc time)
train samples:      (50, 184500) (5, 184500)
validation samples: (50, 20500) (5, 20500)
```

```
begin
    y_test = copy(labels_test)
    y_test = Flux.onehotbatch(y_test, 1:5);

    @time tokens_test = map(d->preprocess(d, tokenizer, max_length=max_length), documents_test)
    @time indices_test = indexer(tokens_test)

    X_test = indices_test

    println("test indices: ", size(indices_test))
    println("test samples: ", size(X_test), " ", size(y_test))
end
```

```
  0.145216 seconds (718.85 k allocations: 46.182 MiB, 29.92% compilation time)
  0.530054 seconds (19 allocations: 1.946 MiB, 0.60% compilation time)
test indices: (50, 5000)
test samples: (50, 5000) (5, 5000)
```

Create the training and validation data loaders

```
321-element DataLoader(::Tuple{Matrix{Int64}, OneHotArrays.OneHotMatrix{UInt32, Vector{UIn
  with first element:
  (50×64 Matrix{Int64}, 5×64 OneHotMatrix(::Vector{UInt32}) with eltype Bool,)
```

```
begin
    train_data_loader = DataLoader(train_data; batchsize=64, shuffle=false);
    val_data_loader  = DataLoader(val_data; batchsize=64, shuffle=false);
end
```

Evaluate

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

```
begin
    loss(x, y) = Flux.logitcrossentropy(model(x), y)
    loss(x::Tuple) = loss(x[1], x[2])
    accuracy(ŷ, y) = mean(Flux.onecold(ŷ) .== Flux.onecold(y))
end
```

```
0.5425365853658537
```

```
@time batched_metric(model, accuracy, train_data_loader)
```

```
169.791909 seconds (7.01 M allocations: 220.288 GiB, 2.60% gc time, 3.22% compilation time)
```

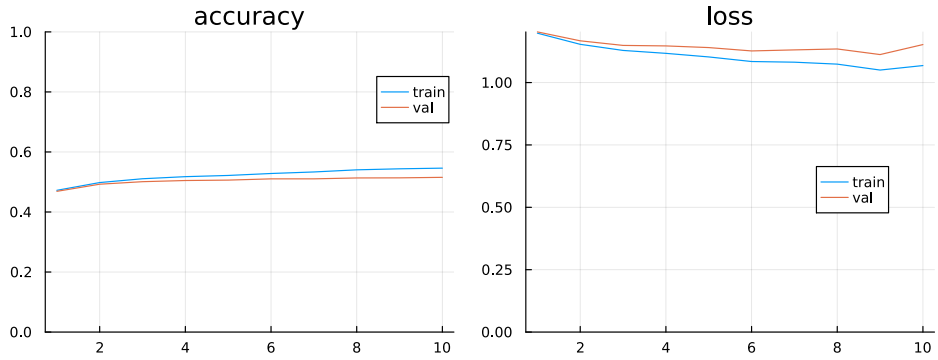
```
0.5474146341463415
```

```
@time batched_metric(model, accuracy, val_data_loader)
```

```
 17.675820 seconds (138.76 k allocations: 24.435 GiB, 2.34% gc time)
```

```
history =
  Dict("train_loss" => [1.19794, 1.15325, 1.12878, 1.11709, 1.10272, 1.08433, 1.08166, 1.073
```

```
history = open(joinpath(directory, "history.json"), "r") do f
    JSON.parse(read(f, String))
end
```




```
begin
    epochs = 1:length(history["train_acc"])
    p1 = plot(epochs, history["train_acc"], label="train")
    plot!(p1, epochs, history["val_acc"], label="val")
    plot!(p1, ylims=[0, 1], title="accuracy", legend=(0.9, 0.8))

    p2 = plot(epochs, history["train_loss"], label="train")
    plot!(p2, epochs, history["val_loss"], label="val")
    plot!(p2, title="loss", ylims=[0, Inf], legend=(0.8, 0.5))

    p3 = plot(p1, p2, layout=grid(1, 2), size=(800, 300))
    savefig(p3, joinpath(directory, "history.png"))
    p3
end
```

Test data

```
0.515
```

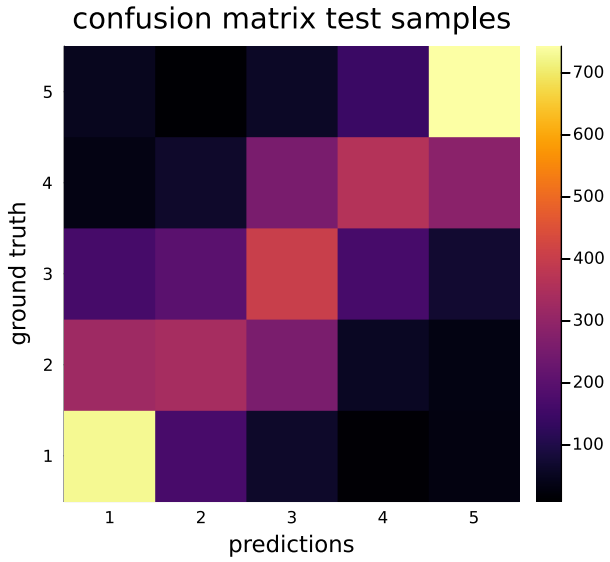
```
begin
    logits = model(X_test)
    accuracy(logits, y_test)
end
```

```
[1, 1, 1, 1, 3, 1, 3, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 2, 2, 2,    more ,3, 5, 1, 5, 1, 4, 5, 5,
```

```
begin
    probs = softmax(logits, dims=1)
    y_pred = Flux.onecold(probs);
end
```

```
cm = 5×5 Matrix{Int64}:
 729  167   63   12   29
 322  337  256   54   31
 163  198  403  165   71
  33   64  255  363  285
  48    9   57  143  743
```

```
cm = confusion_matrix(vec(y_pred), Flux.onecold(y_test), 1:nlabels)
```




```
begin
    p4 = heatmap(1:5, 1:5, cm, xlabel="predictions", ylabel="ground truth", xlims=(0.5, nlabels+0.5), aspectratio=1,
        title="confusion matrix test samples", xticks=(1:5)) #, ["negative", "mix", "positive"]))
    savefig(p4, joinpath(directory, "confusion_matrix.png"))
    p4
end
```

```
classification_report(cm, 1:nlabels)
```

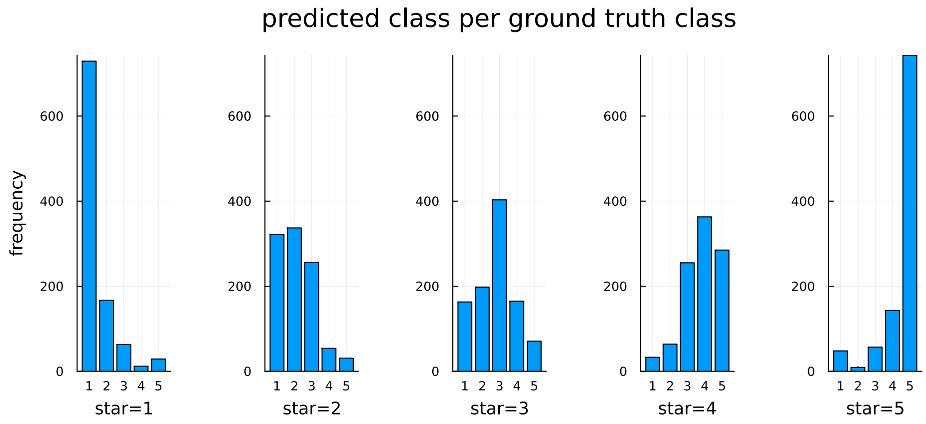
```
              precision  recall  f1-score  support
           1       0.56    0.73      0.64     1000
           2       0.43    0.34      0.38     1000
           3       0.39    0.40      0.40     1000
           4       0.49    0.36      0.42     1000
           5       0.64    0.74      0.69     1000

weighted avg       0.50    0.52      0.50     5000
```

Examples

```
begin
    println("star  y  ŷ   prob")
    for star in nlabels:-1:1
        pos_max = argmax(probs[star, :])
        @printf("   %1d  %d  %d  %.4f\n %s\n\n",
                star, labels_test[pos_max], y_pred[pos_max], probs[star, pos_max], documents_test[pos_max]
            )
    end
end
```

```
star  y  ŷ   prob
   5  5  5  0.9729
 Best purchase I have ever made!!! This mirror is amazing! I highly recommend this product super sturdy, lighting is perfect, super cute and easy to use! I'm recommending this to all my co workers! I'm honestly thinking about buying a second one for myself and a couple for Christmas gifts (:

   4  4  4  0.8998
 Great product. Only complaint I have was in packaging some paints did leak. Otherwise excellent!

   3  3  3  0.7588
 Not good for small to average cup sizes or long torsos. I felt like it was pulling down on my chest. Otherwise good.

   2  3  2  0.6044
 While these worked pretty well, the left earbud broke after less than five months. I got a lovely does of piercingly loud static and high pitched noises during the middle of a workout and then the left one completely died. Disappointed. Update - supplier provided a new set of head phones to replace the defective ones. I've been using them for about a month with no issues. Will update again later if needed.

   1  1  1  0.9838
 Scam, never came in!
```

Probabilities




```
begin
    canvases1 = []
    label_names = 1:5
    for gt_star in 1:5
        idxs = labels_test .== gt_star
        value_counts = [sum((y_pred[idxs]) .== l) for l in 1:nlabels]
        p = bar(value_counts, xlabel="star=$gt_star",legend=:none, xticks=
    (1:nlabels, 1:5))#["neg", "mix", "pos"]))
        push!(canvases1, p)
    end
    plot!(canvases1[1], ylabel="frequency")
    p5 =plot(canvases1..., layout=(1, 5), link=:y, size=(900, 400),
    plot_title="predicted class per ground truth class",
        margin=5Plots.mm)
    savefig(p5, joinpath(directory, "prediction_star.png"))
    p5
end
```

Single sample

```
begin
    idx = 4600

    d = documents_test[idx]
    println(labels_test[idx])
    println(d)
    println("")

    tokens2 = preprocess(d, tokenizer, max_length=50)
    println(join(tokens2, "|"))
    println("")

    x = indexer(tokens2)
    x = vcat(x, ones(Int, 50 - length(x)))
    println(join(x, "|"))
end
```

```
5
I used for a party and was great !

used|for|party|and|was|great

89|7|764|4|14|29|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|1|1|1|1|1|1|1|1|1|1|1|1
```

```
5×1 Matrix{Float32}:
 3.8902854f-5
 0.00019455027
 0.0019970224
 0.14828141
 0.8494881
```

```
softmax(model(x))
```

Evaluation

I was not able to improve much from the baseline.

- When I tried adding an encoder block or more attentions head (trying 6 or 8) the accuracy and validation would not improve
- As mentioned TransformerClassifier() parameters follow a strict order, it appears more dense layers or dropout layers cannot be added with an error
- I got maybe a half a percent improvement by
- using Byte Pair Encoding
- mapping the output of the transformer encoder to 64-dimensional space
- Using the relu acitvation function, which is simple to implement using flux