



Table of Contents

Sentiment analysis using a transformer

Model training

Download module from GitHub

Setup the training data

Tokenisers

Tokenise

Model definition

Training

Save the model

Sentiment analysis using a transformer

Model training

Tip

Hidden below is a useful snippet of HTML to setup a `restart` button in case training gets out of hand.

`Restart`

Download module from GitHub

Because we're working from a GitHub repo and not the standard Julia repository, we have to manage the installation and use of all packages rather than rely on Pluto.

```
1 begin
2     import Pkg
3     Pkg.activate(mktempdir())
4     Pkg.develop(url="https://github.com/rgreilly/Transformers")
5     Pkg.add(["Revise", "PlutoUI", "Flux", "DataFrames", "Printf",
6             "BSON", "JSON", "Arrow", "StatsBase", "Unicode", "Random",
7             "DataStructures", "ProgressMeter", "RemoteFiles"])
8
9     using Revise
10    using TransformersLite
11    using PlutoUI
12    using Flux
13    using Flux.CUDA
14    using Flux: DataLoader
15    using DataFrames
16    using BSON, JSON
17    using Arrow
18    using Printf
19    using StatsBase
20    using StatsBase: mean
21    using Dates
22    using Unicode
23    using Random
24    using DataStructures
25    using RemoteFiles
26 end;
```

```
Activating new project at 'C:\Users\u210148\AppData\Local\Temp\jl_jUW0w3' ⓘ
  Cloning git-repo 'https://github.com/rgreilly/Transformers'
Path 'C:\Users\u210148\.julia\dev\TransformersLite' exists and looks like the
correct repo. Using existing path.
  Resolving package versions...
  Updating 'C:\Users\u210148\AppData\Local\Temp\jl_jUW0w3\Project.toml'
[6579f8b0] + TransformersLite v0.1.0 'C:\Users\u210148\.julia\dev\Transforme
rsLite'
    Updating 'C:\Users\u210148\AppData\Local\Temp\jl_jUW0w3\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] + BFloating16s v0.4.2
[fb218c0] + 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 v4.11.0
[a33af91c] + CompositionsBase v0.1.2
```

In addition to the list of modules, we also need to include individual Julia files from the repo. This is done using the `RemoteFiles` module. However, this downloads them as `JSON` objects, which we need to convert back to regular `.jl` files.

```
1 begin
2     @RemoteFileSet FILES "Transformer utilities" begin
3
4         utilities = @RemoteFile "https://github.com/rgreilly/Transformers/blob/main
5             /examples/utilities.jl" dir="utilities" file="utilities.jl.json"
6
7         training = @RemoteFile "https://github.com/rgreilly/Transformers/blob/main
8             /examples/training.jl" dir="utilities" file="training.jl.json"
9     end
10
11     download(FILES) # Files downloaded in JSON format
12 end
```

`convertJSON` (generic function with 1 method)

```
1 function convertJSON(inFile, outFile)
2     body = JSON.parsefile(inFile)[ "payload" ][ "blob" ][ "rawLines" ]
3     open(outFile, "w") do f
4         for i in body
5             println(f, i)
6         end
7     end
8 end
```

```
1 begin
2     convertJSON("utilities/utilities.jl.json", "utilities/utilities.jl")
3     convertJSON("utilities/training.jl.json", "utilities/training.jl")
4
5     include("utilities/utilities.jl")
6     include("utilities/training.jl")
7 end;
```

Setup the training data

- Setup the file path to the Kaggle Amazon reviews dataset
- Assign values to various hyper-parameters and store them in a dictionary .
- Set number of training epochs

Tip

Here is where you can manipulate various training parameters - pdrop : proportion of weights to drop out (i.e., set to zero); dim-embedding : size of embedding; n_epoch : number of epochs.

```

1 begin
2     path = normpath(joinpath(@__DIR__, "..", "CS401Assignment6/datasets",
3         "amazon_reviews_multi", "en", "1.0.0"))
4     filename = "train.arrow"
5     to_device = cpu # gpu or cpu
6
7     filepath = joinpath(path, filename)
8
9     df = DataFrame(Arrow.Table(filepath))
10    display(first(df, 20))
11    println("")
12
13    hyperparameters = Dict(
14        "seed" => 314159,
15        "tokenizer" => "none", # options: none bpe affixes (why though)
16        "nlabels" => 5,
17        "pdrop" => 0.30,
18        "dim_embedding" => 20
19    )
20    nlabels = hyperparameters["nlabels"]
21    n_epochs = 10
22 end;

```

20x9 DataFrame						
Row	Column1	review_id	product_id	reviewer_id	star ...	(?)
	Int64	String	String	String	Int64 ...	
1	200000	en_0964290	product_en_0740675	reviewer_en_0342986	...	
2	200001	en_0690095	product_en_0440378	reviewer_en_0133349		
3	200002	en_0311558	product_en_0399702	reviewer_en_0152034		
4	200003	en_0044972	product_en_0444063	reviewer_en_0656967		
5	200004	en_0784379	product_en_0139353	reviewer_en_0757638	...	
:	:	:	:	:	:	
17	200016	en_0619473	product_en_0250211	reviewer_en_0056679		
18	200017	en_0533035	product_en_0566399	reviewer_en_0488191		
19	200018	en_0832890	product_en_0304984	reviewer_en_0667005		
20	200019	en_0550306	product_en_0387159	reviewer_en_0627216	...	
5 columns and 11 rows omitted						

Tokenisers

Select a tokeniser. In this case, `none`, which just uses the various inflected word forms.

```
1 begin
2     if hyperparameters["tokenizer"] == "bpe"
3         directory = joinpath("vocab", "bpe")
4         path_rules = joinpath(directory, "amazon_reviews_train_en_rules.txt")
5         path_vocab = joinpath(directory, "amazon_reviews_train_en_vocab.txt")
6         tokenizer = load_bpe(path_rules, startsym=".")
7     elseif hyperparameters["tokenizer"] == "affixes"
8         directory = joinpath("vocab", "affixes")
9         path_vocab = joinpath(directory, "amazon_reviews_train_en_vocab.txt")
10        tokenizer = load_affix_tokenizer(path_vocab)
11    elseif hyperparameters["tokenizer"] == "none"
12        path_vocab = joinpath("vocab", "amazon_reviews_train_en.txt")
13        tokenizer = identity
14    end
15
16    vocab = load_vocab(joinpath(@__DIR__, path_vocab))
17    indexer = IndexTokenizer(vocab, "[UNK]")
18
19    display(tokenizer)
20    println("")
21    display(indexer)
22    println("")
23
24 end
```

identity (generic function with 1 method)

?

IndexTokenizer{String}(length(vocabulary)=6654, unksym=[UNK])

Tokenise

Extract the review body and star rating from the dataframe and create embeddings. Partition data into training and validation sets.

```
1 begin
2     documents = df[!, :review_body]
3     labels = df[!, :stars]
4     max_length = 50
5     indices_path = joinpath(@__DIR__, "outputs", "indices_" *
6         hyperparameters["tokenizer"] * ".bson")
7     @time tokens = map(d->preprocess(d, tokenizer, max_length=max_length),
8         documents)
9     @time indices = indexer(tokens)
10
11    y_labels = Int.(labels)
12    if nlabels == 1
13        y_labels[labels .≤ 2] .= 0
14        y_labels[labels .≥ 4] .= 1
15        idxs = labels .!= 3
16        y_labels = reshape(y_labels, 1, :)
17    else
18        idxs = Base.OneTo(length(labels))
19        y_labels = Flux.onehotbatch(y_labels, 1:nlabels)
20    end
21
22    X_train, y_train = indices[:, idxs], y_labels[:, idxs];
23    rng = MersenneTwister(hyperparameters["seed"])
24    train_data, val_data = split_validation(X_train, y_train; rng=rng)
25
26    println("train samples:      ", size(train_data[1]), " ", size(train_data[2]))
27    println("validation samples: ", size(val_data[1]), " ", size(val_data[2]))
28    println("")
end
```

```
4.501375 seconds (28.85 M allocations: 1.836 GiB, 18.22% gc time, 3.12% compilation time)
15.712033 seconds (31.81 k allocations: 81.911 MiB, 0.14% gc time, 0.29% compilation time)
train samples:      (50, 184500) (5, 184500)
validation samples: (50, 20500) (5, 20500)
```

Model definition

Assemble the model's components.

Tip

Here's where you might want to adjust the number and nature of the encoder blocks (e.g., attention heads, dropout), number of Dense layers and their characteristics (e.g., activation function, dimensions), the number of dropout layers.

```
1 begin
2     dim_embedding = hyperparameters["dim_embedding"]
3     pdrop = hyperparameters["pdrop"]
4     model = TransformersLite.TransformerClassifier(
5         Embed(dim_embedding, length(indexer)),
6         PositionEncoding(dim_embedding),
7         Dropout(pdrop),
8         TransformerEncoderBlock[
9             TransformerEncoderBlock(4, dim_embedding, dim_embedding * 4;
10                pdrop=pdrop)
11            ],
12            Dense(dim_embedding, 1),
13            FlattenLayer(),
14            Dense(max_length, nlabels)
15        )
16    display(model)
17    println("")
18    model = to_device(model)
19
20    hyperparameters["model"] = "$(typeof(model).name.wrapper)"
21    hyperparameters["trainable parameters"] = sum(length, Flux.params(model));
22
23    if nlabels == 1
24        loss(x, y) = Flux.logitbinarycrossentropy(x, y)
25        accuracy(ŷ, y) = mean((Flux.sigmoid.(ŷ) .> 0.5) .== y)
26    else
27        loss(x, y) = Flux.logitcrossentropy(x, y)
28        accuracy(ŷ, y) = mean(Flux.onecold(ŷ) .== Flux.onecold(y))
29    end
30
31 end;
```

```
TransformerClassifier(
    Embed((20, 6654)),                                # 133_080 parameters
    PositionEncoding(20),
    Dropout(0.3),
    TransformerEncoderBlock(
        MultiheadAttention(num_heads=4, head_size=5, 20=>20)(
            denseQ = Dense(20 => 20),                 # 420 parameters
            denseK = Dense(20 => 20),                 # 420 parameters
            denseV = Dense(20 => 20),                 # 420 parameters
            denseO = Dense(20 => 20),                 # 420 parameters
        ),
        Dropout(0.3),
        LayerNorm(20),                                # 40 parameters
        Dense(20 => 80, relu),                         # 1_680 parameters
        Dense(80 => 20),                                # 1_620 parameters
        Dropout(0.3),
        LayerNorm(20),                                # 40 parameters
    ),
    Dense(20 => 1),                                 # 21 parameters
    FlattenLayer(),
    Dense(50 => 5),                                # 255 parameters
)      # Total: 21 trainable arrays, 138_416 parameters,
      # plus 1 non-trainable, 20_000 parameters, summarysize 620.320 KiB.
```

Training

- Setup the dataloaders to batch and shuffle the training and validation data.
- Print out initial accuracy and loss values for the validation data.
- Setup a sub-directory in the outputs directory, based on date and time, to store the trained model and associated hyperparameters.
- call the `train!` method and log training progress.

```
1 begin
2     opt_state = Flux.setup(Adam(), model)
3     batch_size = 32
4
5     train_data_loader = DataLoader(train_data |> to_device; batchsize=batch_size,
6         shuffle=true)
7     val_data_loader = DataLoader(val_data |> to_device; batchsize=batch_size,
8         shuffle=false)
9
10    val_acc = batched_metric(model, accuracy, val_data_loader)
11    val_loss = batched_metric(model, loss, val_data_loader)
12
13    @printf "val_acc=% .4f% ; " val_acc * 100
14    @printf "val_loss=% .4f \n" val_loss
15    println("")
16
17    directory2 = normpath(joinpath(@__DIR__, "..", "outputs",
18        Dates.format(now(), "yyyymmdd_HHMM")))
19    mkpath(directory2)
20    output_path = joinpath(directory2, "model.bson")
21    history_path = joinpath(directory2, "history.json")
22
23    hyperparameter_path = joinpath(directory2, "hyperparameters.json")
24    open(hyperparameter_path, "w") do f
25        JSON.print(f, hyperparameters)
26    end
27    println("saved hyperparameters to $(hyperparameter_path).")
28    println("")
29
30    start_time = time_ns()
31    history = train!(
32        loss, model, train_data_loader, opt_state, val_data_loader;
33        num_epochs=n_epochs)
34    end_time = time_ns() - start_time
35
36    println("done training")
37    @printf "time taken: %.2fs\n" end_time/1e9
38 end
```

```
val_acc=19.3951% ; val_loss=2.3643
```

?

```
saved hyperparameters to C:\Users\u210148\Documents\outputs\20240111_1629\hyperparameters.json.
```

```
epoch 1/10  0%|██████████| 0%  ETA: 22:09:51
mean_loss:  2.296135425567627
batch_loss: 2.3022406
```

```
epoch 1/10  1%|████| 1%  ETA: 1:09:56
mean_loss:  2.01741270835583
```

```
batch_loss: 1.6442395
```

```
epoch 1/10 1%|██████████| mean_loss: 1.9523112567571492  
batch_loss: 1.775497
```

| ETA: 0:52:42

accuracy (generic function with 1 method)

```
1 accuracy
```

Save the model

Save model, embeddings, and training history to the outputs sub-directory.

```
1 begin  
2     model2 = model |> cpu  
3     if hasproperty(tokenizer, :cache)  
4         # empty cache  
5         tokenizer2 = similar(tokenizer)  
6     end  
7     BSON.bson(  
8         output_path,  
9         Dict(  
10            :model=> model2,  
11            :tokenizer=>tokenizer,  
12            :indexer=>indexer  
13        ))  
14    )  
15    println("saved model to $(output_path).")  
16  
17    open(history_path, "w") do f  
18        JSON.print(f, history)  
19    end  
20    println("saved history to $(history_path).")  
21  
22 end
```

```
saved model to C:\Users\u210148\Documents\outputs\20240111_1629\model.bson. ⓘ  
saved history to C:\Users\u210148\Documents\outputs\20240111_1629\history.json.
```

Tip

Take note of the timestamped sub-directory so that you can load the saved model and parameters for use in the evaluation notebook.

```
1 md"""
2 !!! tip
3     Take note of the timestamped sub-directory so that you can load the saved model
        and parameters for use in the evaluation notebook.
4 """
```


