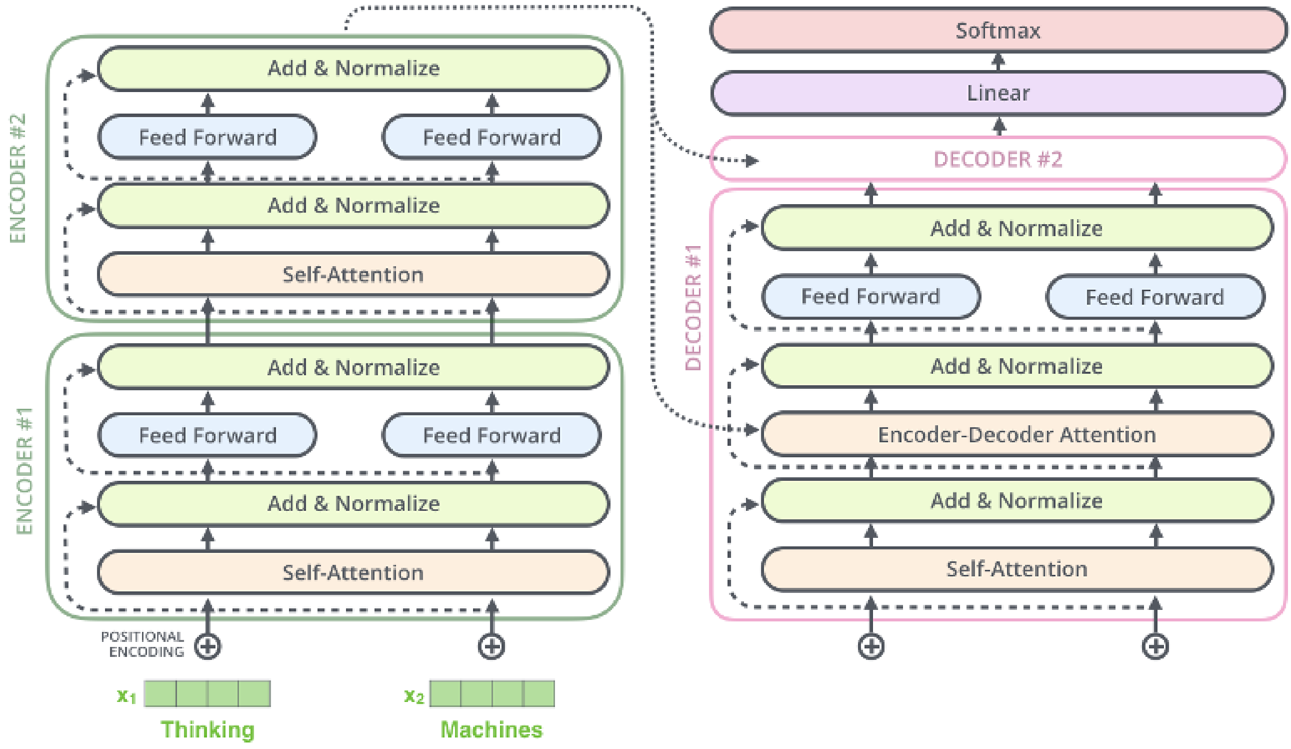

Table of Contents

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.

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

    using Revise
    using TransformersLite
    using PlutoUI
    using Flux
    using Flux.CUDA
    using Flux: DataLoader
    using DataFrames
    using BSON, JSON
    using Arrow
    using Printf
    using StatsBase
    using StatsBase: mean
    using Dates
    using Unicode
    using Random
    using DataStructures
    using TransformersLite
    using RemoteFiles
end;
```

```
  Activating new project at `C:\Users\alexm\AppData\Local\Temp\jl_BQHHFa
`
     Cloning git-repo `https://github.com/rgreilly/Transformers`
Path `C:\Users\alexm\.julia\dev\TransformersLite` exists and looks like the
correct repo. Using existing path.
   Resolving package versions...
    Updating `C:\Users\alexm\AppData\Local\Temp\jl_BQHHFa\Project.toml`
  [6579f8b0] + TransformersLite v0.1.0 `C:\Users\alexm\.julia\dev\Transforme
rsLite`
    Updating `C:\Users\alexm\AppData\Local\Temp\jl_BQHHFa\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
```

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.

```julia
begin
    @RemoteFileSet FILES "Transformer utilities" begin

        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)
```

```julia
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
```

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

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

Setup the training data

- Setup the file path to the Kaggle Amazon reviews dataset
- Assign values to various hyper-paraemeters 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 dropout (i.e., set to zero); `dim-embedding`: size of embedding; `n_epoch`: number of epochs.

```
begin
    path = normpath(joinpath(@__DIR__, "..", "assignment6/datasets",
    "amazon_reviews_multi", "en", "1.0.0"))
    filename = "train.arrow"
    to_device = cpu # gpu or cpu

    filepath = joinpath(path, filename)

    df = DataFrame(Arrow.Table(filepath))
    display(first(df, 20))
    println("")

    hyperparameters = Dict(
        "seed" => 314159,
        "tokenizer" => "none", # options: none bpe affixes
        "nlabels" => 5,
        "pdrop" => 0.75, #CHANGE HERE FROM 0.1->0.5->0.75
        "dim_embedding" => 128 #CHANGE HERE FROM 32->64->128
    )
    nlabels = hyperparameters["nlabels"]
    n_epochs = 10
end;
```

```
20×9 DataFrame
 Row │ Column1  review_id   product_id           reviewer_id           star ⋯
     │ Int64    String      String               String                Int6 ⋯
─────┼──────────────────────────────────────────────────────────────────────────
   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.

```julia
begin
    if hyperparameters["tokenizer"] == "bpe"
        directory = joinpath("vocab", "bpe")
        path_rules = joinpath(directory, "amazon_reviews_train_en_rules.txt")
        path_vocab = joinpath(directory, "amazon_reviews_train_en_vocab.txt")
        tokenizer = load_bpe(path_rules, startsym="·")
    elseif hyperparameters["tokenizer"] == "affixes"
        directory = joinpath("vocab","affixes")
        path_vocab = joinpath(directory, "amazon_reviews_train_en_vocab.txt")
        tokenizer = load_affix_tokenizer(path_vocab)
    elseif hyperparameters["tokenizer"] == "none"
        path_vocab = joinpath("vocab", "amazon_reviews_train_en.txt")
        tokenizer = identity
    end

    vocab = load_vocab(joinpath(@__DIR__, path_vocab))
    indexer = IndexTokenizer(vocab, "[UNK]")

    display(tokenizer)
    println("")
    display(indexer)
    println("")

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.

```
begin
    documents = df[!, :review_body]
    labels = df[!, :stars]
    max_length = 50
    indices_path = joinpath(@__DIR__, "outputs", "indices_" *
    hyperparameters["tokenizer"] * ".bson")
    @time tokens = map(d->preprocess(d, tokenizer, max_length=max_length),
    documents)
    @time indices = indexer(tokens)

    y_labels = Int.(labels)
    if nlabels == 1
        y_labels[labels .≤ 2] .= 0
        y_labels[labels .≥ 4] .= 1
        idxs = labels .!= 3
        y_labels = reshape(y_labels, 1, :)
    else
        idxs = Base.OneTo(length(labels))
        y_labels = Flux.onehotbatch(y_labels, 1:nlabels)
    end

    X_train, y_train = indices[:, idxs], y_labels[:, idxs];
    rng = MersenneTwister(hyperparameters["seed"])
    train_data, val_data = split_validation(X_train, y_train; rng=rng)

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

```
  4.023848 seconds (28.85 M allocations: 1.836 GiB, 17.55% gc time, 3.22% c
ompilation time)
 12.477690 seconds (31.81 k allocations: 81.916 MiB, 0.33% gc time, 0.53% comp
ilation 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.

```
MethodError: no method matching TransformersLite.TransformerEncoderBlock(::Int64, ::Int64, ::Int64; nheads::Int64, pdrop::Float64)
Closest candidates are:
TransformersLite.TransformerEncoderBlock(::Int64, ::Int64, ::Int64; pdrop, act) got unsupported keyword argument "nheads"
@ TransformersLite C:\Users\alexm\.julia\dev\TransformersLite\src\encoder.jl:21
```

1. **kwerr**(::NamedTuple{(:nheads, :pdrop), Tuple{Int64, Float64}}, ::Type, ::Int64, ::Int64, ::Int64) @ *error.jl:165*
2. **top-level scope** @ *Local: 4*

```julia
begin
    dim_embedding = hyperparameters["dim_embedding"]
    pdrop = hyperparameters["pdrop"]
    model = TransformersLite.TransformerClassifier(
        Embed(dim_embedding, length(indexer)),
        PositionEncoding(dim_embedding),
        Dropout(pdrop),
        TransformerEncoderBlock[
            TransformerEncoderBlock(4, dim_embedding, dim_embedding * 4; pdrop=pdrop), #Added ,

            TransformerEncoderBlock(4, dim_embedding, dim_embedding * 4; pdrop=pdrop), #Added extra transformer block normal

                TransformerEncoderBlock(4, dim_embedding, dim_embedding * 4, nheads=8; pdrop=pdrop), #Added extra encoder block with attention Head
        ],
        Dropout(pdrop), #Adding Dropout Layers
        Dense(dim_embedding, 1),
        FlattenLayer(),
        Dense(max_length, nlabels),
        Dropout(pdrop), #Adding Dropout Layers
        Dense(dim_embedding, 256, relu), #Adding dense layers
        Dropout(pdrop), #Adding Dropout Layers
        Dense(256, 128, relu), #Adding dense layers
        )
    display(model)
    println("")
    model = to_device(model)

    hyperparameters["model"] = "$(typeof(model).name.wrapper)"
    hyperparameters["trainable parameters"] = sum(length, Flux.params(model));

    if nlabels == 1
        loss(x, y) = Flux.logitbinarycrossentropy(x, y)
        accuracy(ŷ, y) = mean((Flux.sigmoid.(ŷ) .> 0.5) .== y)
    else
        loss(x, y) = Flux.logitcrossentropy(x, y)
        accuracy(ŷ, y) = mean(Flux.onecold(ŷ) .== Flux.onecold(y))
    end
end;
```

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.

```
UndefVarError: `model` not defined
```

```julia
begin
    opt_state = Flux.setup(Adam(), model)
    batch_size = 32

    train_data_loader = DataLoader(train_data |> to_device; batchsize=batch_size,
        shuffle=true)
    val_data_loader = DataLoader(val_data |> to_device; batchsize=batch_size,
        shuffle=false)

    val_acc = batched_metric(model, accuracy, val_data_loader)
    val_loss = batched_metric(model, loss, val_data_loader)

    @printf "val_acc=%.4f%% ; " val_acc * 100
    @printf "val_loss=%.4f \n" val_loss
    println("")

    directory2 = normpath( joinpath(@__DIR__, "..", "outputs",
        Dates.format(now(), "yyyymmdd_HHMM")))
    mkpath(directory2)

    hyperparameter_path = joinpath(directory2, "hyperparameters.json")
    open(hyperparameter_path, "w") do f
        JSON.print(f, hyperparameters)
    end
    println("saved hyperparameters to $(hyperparameter_path).")
    println("")

    start_time = time_ns()
    history = train!(
        loss, model, train_data_loader, opt_state, val_data_loader;
            num_epochs=n_epochs)
    end_time = time_ns() - start_time

    println("done training")
    @printf "time taken: %.2fs\n" end_time/1e9
end
```

```
UndefVarError: `accuracy` not defined
```

```julia
accuracy
```

Save the model

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

```
UndefVarError: `model` not defined
```

```julia
begin
    model2 = model |> cpu
    if hasproperty(tokenizer, :cache)
        # empty cache
        tokenizer2 = similar(tokenizer)
    end
    output_path = joinpath(directory2, "model.bson")
    history_path = joinpath(directory2, "history.json")
    BSON.bson(
        output_path,
        Dict(
            :model=> model2,
            :tokenizer=>tokenizer,
            :indexer=>indexer
        )
    )
    println("saved model to $(output_path).")

    open(history_path,"w") do f
      JSON.print(f, history)
    end
    println("saved history to $(history_path).")

end
```

Tip

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