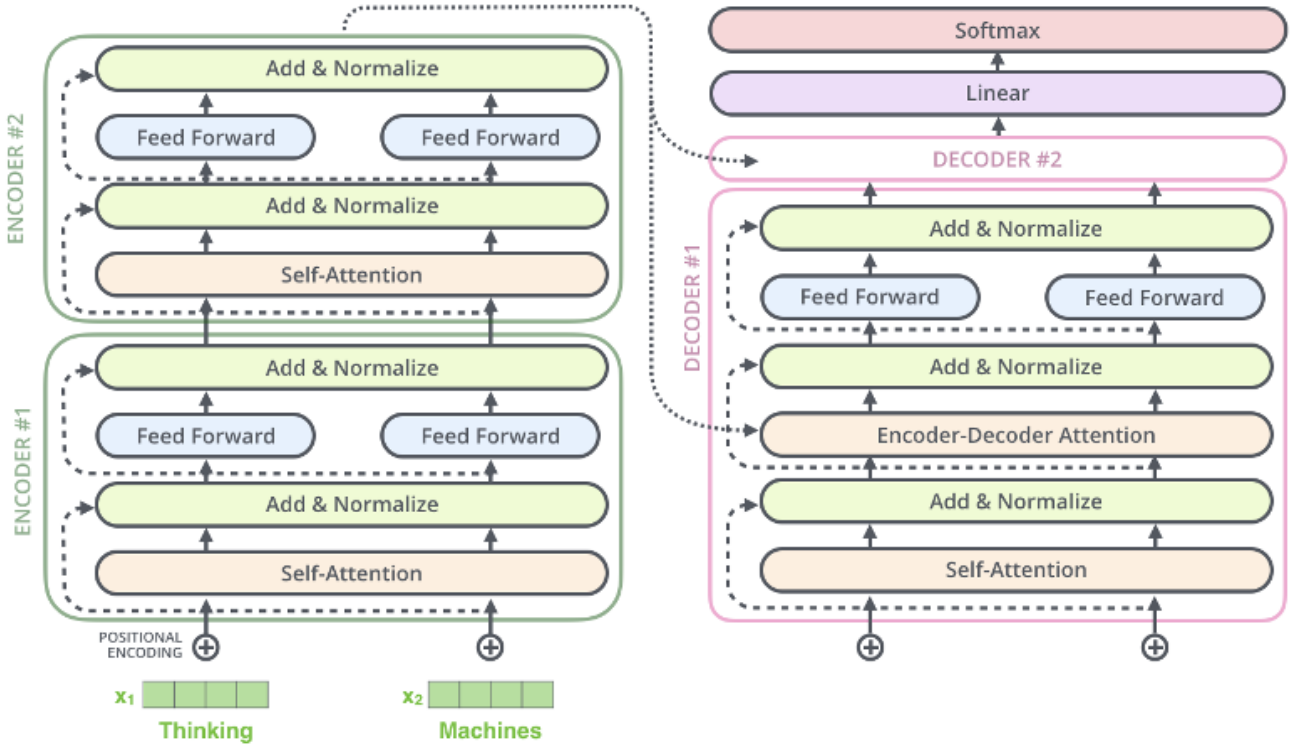


Sentiment analysis using a transformer

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
end;
```

```
  Activating new project at `C:\Users\u210148\AppData\Local\Temp\jl_9Sqjzt`
     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_9Sqjzt\Project.toml`
  [6579f8b0] + TransformersLite v0.1.0 `C:\Users\u210148\.julia\dev\Transforme
rsLite`
    Updating `C:\Users\u210148\AppData\Local\Temp\jl_9Sqjzt\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 v4.11.0
  [a33af91c] + CompositionsBase v0.1.2
```

```
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
    directory = "../outputs/20240111_1629/"
    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((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 620.500 KiB.
```

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
```

```
  4.467094 seconds (28.48 M allocations: 1.786 GiB, 15.92% gc time)
 14.673765 seconds (4 allocations: 79.765 MiB)
train samples:      (50, 184500) (5, 184500)
validation samples: (50, 20500) (5, 20500)
```

Tokenise

```
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
```

```
  0.114423 seconds (718.85 k allocations: 46.182 MiB, 26.68% compilation time)
  0.357098 seconds (19 allocations: 1.946 MiB, 1.00% 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{UI
  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
```

```
0.5246124661246613
```

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

```
 37.649349 seconds (6.50 M allocations: 43.466 GiB, 7.74% gc time, 10.66% c
ompilation time)
```

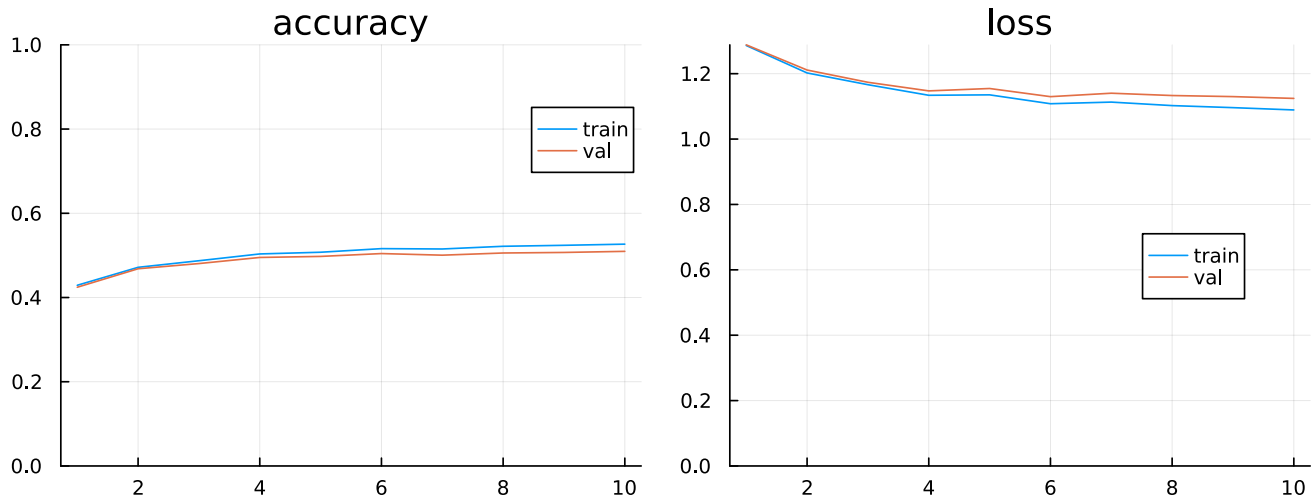
```
0.5284390243902439
```

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

```
  3.564160 seconds (80.86 k allocations: 4.788 GiB, 8.48% gc time)
```

```
history =
  Dict("train_loss" ⇒ [1.2862, 1.20225, 1.16633, 1.13392, 1.13527, 1.10809, 1.11309, 1.10
```

```
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

```
0.5104
```

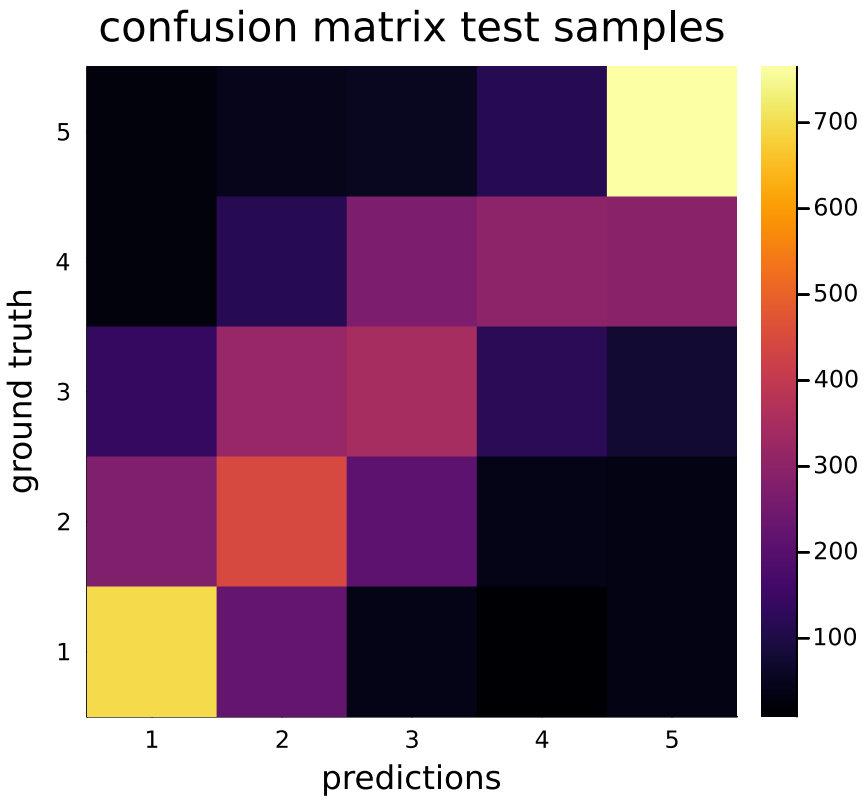
```
begin
    logits = model(X_test)
    accuracy(logits, y_test)
end
```

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

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

```
cm = 5×5 Matrix{Int64}:
 697  222   38    9   34
 276  443  208   39   34
 140  318  347  121   74
  24  115  267  300  294
  26   43   51  115  765
```

```
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)
```

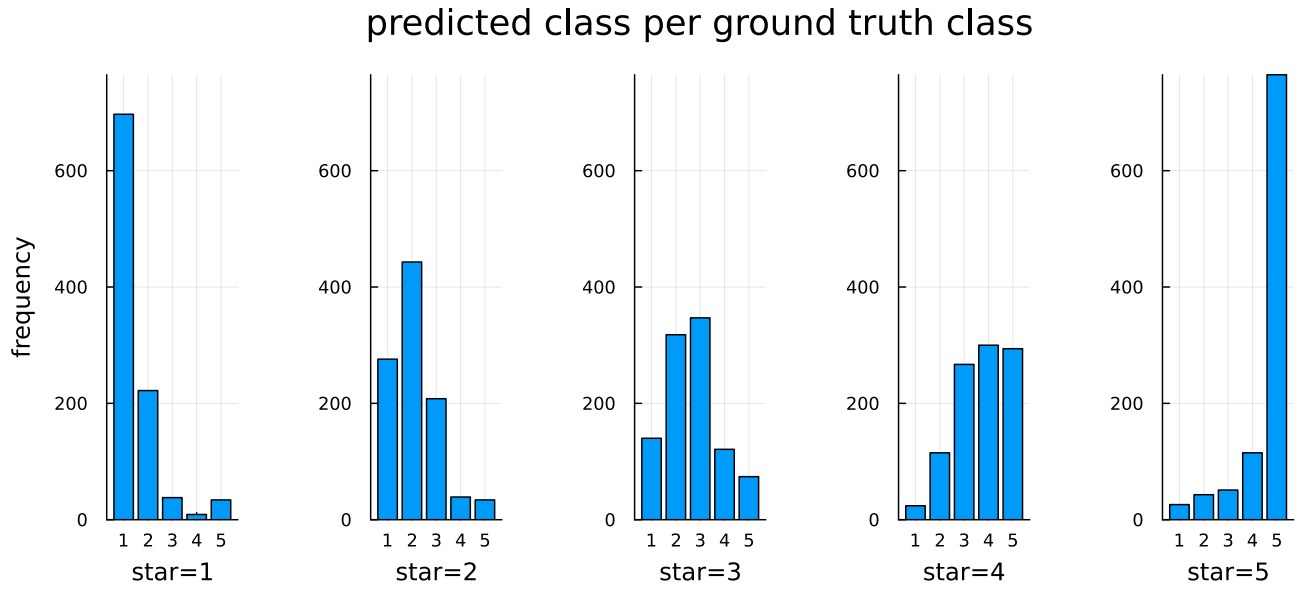
	precision	recall	f1-score	support
1	0.60	0.70	0.64	1000
2	0.39	0.44	0.41	1000
3	0.38	0.35	0.36	1000
4	0.51	0.30	0.38	1000
5	0.64	0.77	0.70	1000
weighted avg	0.50	0.51	0.50	5000

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  0.9788
 The item ordered came exactly as advertised. I highly recommend this vendor a
nd would order from them again.

   4  4  4  0.8214
 Love the shirt. I wish the fabric of the shirt was softer.... a little rough.
Overall good shirt

   3  2  3  0.6738
 I liked it okay while it lasted... I really loved this case when I bought it.
However it cracked & leaked out. In addition, the lip needs to be a little bit
bigger around the edge of the phone. I've gone through MANY, MANY Bodyguardz s
creen protectors with this case.

   2  3  2  0.5584
 It was great while it lasted, but it started falling apart after a couple of
months. The brown inner lining ripped from the inner seam of the leather and i
t's not really fixable. Not great for long-term every day use.

   1  1  1  0.9749
 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}:
 0.00050479587
 0.0014514992
 0.0063166833
 0.13045634
 0.8612706
```

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

```
# When

# The pdrop is 0.30 as dropping too many words loses context. Setting it high can lead to a lot words lost in the training, which can lead to the meaning of these words losing context. Example: Look at how the context of Great is used differently here: I have an exam. Great. - I got 100% in my exam! Great!
#It is at 0.3 to introduce some level of randomness to the model while still ensuring that there's enougugh Context to work with.

#Overtraining occurs when a model fits the training data a bit too excessively. This compromises its ability to generalize to new data. Regularization prevents this, and is why I've reduced the size to 20.

```