

COMP0090 Coursework 2

Tom Grigg
19151291
tom.grigg.19@ucl.ac.uk

Oliver Slumbers
19027699
oliver.slumbers.19@ucl.ac.uk

Agnieszka Dobrowolska
16034489
zcqsaad@ucl.ac.uk

Charita Dellaporta
19025680
charita.dellaporta.19@ucl.ac.uk

Sibo Zhao
19099708
ucabsz6@ucl.ac.uk

December 2019

1 Hyperlinks

Task 1: https://colab.research.google.com/drive/1Gf4A0mgMI3TKX91F9r8_6QGLmKHvJ62j

Task 2: <https://colab.research.google.com/drive/1W5nRNUlkDpkvxT6muwDsnJGBdhu3L0i->

Task 3: <https://colab.research.google.com/drive/1EhNGSsXst2nbdz01hFiXaufdZfX2Ayxe>

Task 4: <https://colab.research.google.com/drive/1FdhB4TqBo1Y9W0UaQ3LE2dB3HMoJ0Hkp>

Task 5: <https://colab.research.google.com/drive/1XYL6dcVfFWWh2x1jQzuRCiGyxFbpktNbd>

2 Contributions inside the group

For this assignment, initially Tom contributed towards questions 1,5, Oliver contributed towards questions 1 and 3, Charita contributed towards questions 2 and 4, Agnieszka contributed towards questions 1 and 3 and Sibbo contributed

towards questions 4,5. However, after our first implementations we all worked as a group to complete all questions and discuss the model variants for each model.

Julia on Colaboratory

[Colaboratory](#) does not provide native support for the [Julia programming language](#). However, since Colaboratory gives you root access to the machine that runs your notebook (the “runtime” in Colaboratory terminology), we can install Julia support by uploading a specially crafted Julia notebook – *this* notebook. We then install Julia and [IJulia](#) ([Jupyter](#)/Colaboratory notebook support) and reload the notebook so that Colaboratory detects and initiates what we installed.

In brief:

1. **Run the cell below**
2. **Reload the page**
3. **Edit the notebook name and start hacking Julia code below**

If your runtime resets, either manually or if left idle for some time, repeat steps 1 and 2.

Acknowledgements

This hack by Pontus Stenertorp is an adaptation of [James Bradbury’s original Colaboratory Julia hack](#), that broke some time in September 2019 as Colaboratory increased their level of notebook runtime isolation. There also appears to be CUDA compilation support installed by default for each notebook runtime type in October 2019, which shaves off a good 15 minutes or so from the original hack’s installation time.

In [0]:

```
%%shell
if ! command -v julia 2>&1 > /dev/null
then
    wget 'https://julialang-s3.julialang.org/bin/linux/x64/1.0/julia-1.0.5-linux-x86_64.tar.gz' \
        -O /tmp/julia.tar.gz
    tar -x -f /tmp/julia.tar.gz -C /usr/local --strip-components 1
    rm /tmp/julia.tar.gz
fi
julia -e 'using Pkg; pkg"add IJulia; precompile;"'
```

Unrecognized magic `%%shell`.

Julia does not use the IPython `%magic` syntax. To interact with the IJulia kernel, use `IJulia.somefunction(...)`, for example. Julia macros, string macros, and functions can be used to accomplish most of the other functionalities of IPython magics.

In [0]:

```
using Pkg

pkg"add MLDatasets; precompile;"
using MLDatasets

pkg"add ImageMagick; precompile;"
using ImageMagick

pkg"add Images; precompile;"
using Images

pkg"add Embeddings; precompile;"
using Embeddings

pkg"add Plots; precompile;"
using Plots
Pkg.add(Pkg.PackageSpec(;name="CuArrays", version="1.3.0"))
using CuArrays

Pkg.add(Pkg.PackageSpec(;name="Flux", version="0.9.0"))
using Flux

using LinearAlgebra
using Base.Iterators
using Random
using Statistics
```

```
Pkg.add("DataFrames")
using DataFrames
```

```
Pkg.add("ProgressBars")
using ProgressBars
```

```
Updating registry at `~/.julia/registries/General`
Updating git-repo `https://github.com/JuliaRegistries/General.git`
Resolving package versions...
Installed Reexport ───────── v0.2.0
Installed Requires ───────── v1.0.0
Installed URIParser ───────── v0.4.0
Installed GZip ───────── v0.5.1
Installed BinDeps ───────── v1.0.0
Installed DataDeps ───────── v0.7.1
Installed MLDatasets ───────── v0.4.0
Installed ColorTypes ───────── v0.8.0
Installed IniFile ───────── v0.5.0
Installed FixedPointNumbers ─ v0.6.1
Installed HTTP ───────── v0.8.8
Updating `~/.julia/environments/v1.0/Project.toml`
[eb30cadb] + MLDatasets v0.4.0
Updating `~/.julia/environments/v1.0/Manifest.toml`
[9e28174c] + BinDeps v1.0.0
[3da002f7] + ColorTypes v0.8.0
[124859b0] + DataDeps v0.7.1
[53c48c17] + FixedPointNumbers v0.6.1
[92fee26a] + GZip v0.5.1
[cd3eb016] + HTTP v0.8.8
[83e8ac13] + IniFile v0.5.0
[eb30cadb] + MLDatasets v0.4.0
[189a3867] + Reexport v0.2.0
[ae029012] + Requires v1.0.0
[30578b45] + URIParser v0.4.0
Precompiling project...
Precompiling MLDatasets
```

```
└ Info: Precompiling MLDatasets [eb30cadb-4394-5ae3-aed4-317e484a6458]
└ @ Base loading.jl:1192
```

```
Resolving package versions...
Installed NaNMath ─── v0.3.3
Installed MappedArrays ─ v0.2.2
Installed AbstractFFTs ─ v0.5.0
Installed OffsetArrays ─ v0.11.3
Installed Graphics ─── v1.0.0
Installed FFTW ─── v1.1.0
Installed ImageMagick ─ v0.7.5
Installed Colors ─── v0.9.6
Installed PaddedViews ─ v0.4.2
Installed FileIO ─── v1.2.0
Installed ImageCore ─ v0.8.6
Updating `~/.julia/environments/v1.0/Project.toml`
[6218d12a] + ImageMagick v0.7.5
Updating `~/.julia/environments/v1.0/Manifest.toml`
[621f4979] + AbstractFFTs v0.5.0
[5ae59095] + Colors v0.9.6
[7a1cc6ca] + FFTW v1.1.0
[5789e2e9] + FileIO v1.2.0
[a2bd30eb] + Graphics v1.0.0
[a09fc81d] + ImageCore v0.8.6
[6218d12a] + ImageMagick v0.7.5
[dbb5928d] + MappedArrays v0.2.2
[77ba4419] + NaNMath v0.3.3
[6fe1bfb0] + OffsetArrays v0.11.3
[5432bcbf] + PaddedViews v0.4.2
Building FFTW ─────────→ `~/.julia/packages/FFTW/loJ3F/deps/build.log`
Building ImageMagick → `~/.julia/packages/ImageMagick/vMfoS/deps/build.log`
Precompiling project...
Precompiling ImageMagick
```

```
└ Info: Precompiling ImageMagick [6218d12a-5da1-5696-b52f-db25d2ecc6d1]
└ @ Base loading.jl:1192
└ Warning: Replacing docs for `FileIO.filename :: Tuple{Any}` in module `FileIO`
└ @ Base Docs docs/Docs.jl:223
```

```
Warning: Replacing docs for `FileIO.file_extension :: Tuple{Any}` in module `FileIO`  
@ Base.Docs docs/Docs.jl:223
```

```
Resolving package versions...  
Installed ComputationalResources v0.3.0  
Installed ImageMorphology v0.2.4  
Installed ImageAxes v0.6.1  
Installed MacroTools v0.5.3  
Installed DataStructures v0.17.6  
Installed AxisArrays v0.3.3  
Installed Images v0.19.1  
Installed WoodburyMatrices v0.4.1  
Installed ImageDistances v0.2.5  
Installed Rotations v0.12.0  
Installed Interpolations v0.12.5  
Installed SpecialFunctions v0.8.0  
Installed DataAPI v1.1.0  
Installed AxisAlgorithms v1.0.0  
Installed ImageTransformations v0.8.0  
Installed CustomUnitRanges v0.2.0  
Installed CatIndices v0.2.0  
Installed TiledIteration v0.2.3  
Installed Distances v0.8.2  
Installed FFTViews v0.3.0  
Installed IntervalSets v0.3.2  
Installed StaticArrays v0.12.1  
Installed Ratios v0.3.1  
Installed ImageFiltering v0.6.7  
Installed IdentityRanges v0.3.0  
Installed CoordinateTransformations v0.5.0  
Installed RangeArrays v0.3.1  
Installed IndirectArrays v0.5.0  
Installed ImageShow v0.2.0  
Installed ColorVectorSpace v0.7.1  
Installed Missings v0.4.3  
Installed SortingAlgorithms v0.3.1  
Installed SimpleTraits v0.9.1  
Installed StatsBase v0.32.0  
Installed ImageMetadata v0.7.2  
Installed OrderedCollections v1.1.0  
Installed IterTools v1.3.0  
Installed Compat v2.2.0  
Updating `~/ .julia/environments/v1.0/Project.toml`  
[916415d5] + Images v0.19.1  
Updating `~/ .julia/environments/v1.0/Manifest.toml`  
[13072b0f] + AxisAlgorithms v1.0.0  
[39de3d68] + AxisArrays v0.3.3  
[aafaddc9] + CatIndices v0.2.0  
[c3611d14] + ColorVectorSpace v0.7.1  
[34da2185] + Compat v2.2.0  
[ed09eef8] + ComputationalResources v0.3.0  
[150eb455] + CoordinateTransformations v0.5.0  
[dc8bdbbb] + CustomUnitRanges v0.2.0  
[9a962f9c] + DataAPI v1.1.0  
[864edb3b] + DataStructures v0.17.6  
[b4f34e82] + Distances v0.8.2  
[4f61f5a4] + FFTViews v0.3.0  
[bbac6d45] + IdentityRanges v0.3.0  
[2803e5a7] + ImageAxes v0.6.1  
[51556ac3] + ImageDistances v0.2.5  
[6a3955dd] + ImageFiltering v0.6.7  
[bc367c6b] + ImageMetadata v0.7.2  
[787d08f9] + ImageMorphology v0.2.4  
[4e3cecf8] + ImageShow v0.2.0  
[02fcd773] + ImageTransformations v0.8.0  
[916415d5] + Images v0.19.1  
[9b13fd28] + IndirectArrays v0.5.0  
[a98d9a8b] + Interpolations v0.12.5  
[8197267c] + IntervalSets v0.3.2  
[c8e1da08] + IterTools v1.3.0  
[1914dd2f] + MacroTools v0.5.3  
[eld29d7a] + Missings v0.4.3  
[bac558e1] + OrderedCollections v1.1.0  
[b3c3ace0] + RangeArrays v0.3.1  
[c84ed2f1] + Ratios v0.3.1  
[6038ab10] + Rotations v0.12.0
```

```
[699a6c99] + SimpleTraits v0.9.1
[a2af1166] + SortingAlgorithms v0.3.1
[276daf66] + SpecialFunctions v0.8.0
[90137ffa] + StaticArrays v0.12.1
[2913bbd2] + StatsBase v0.32.0
[06e1c1a7] + TiledIteration v0.2.3
[efce3f68] + WoodburyMatrices v0.4.1
[8bb1440f] + DelimitedFiles
[1a1011a3] + SharedArrays
[2f01184e] + SparseArrays
[10745b16] + Statistics
Building SpecialFunctions → `~/julia/packages/SpecialFunctions/ne2iw/deps/build.log`
Precompiling project...
Precompiling Images
```

```
[ Info: Precompiling Images [916415d5-f1e6-5110-898d-aaa5f9f070e0]
└ @ Base loading.jl:1192
WARNING: Method definition _bcs1(Any, Any) in module Broadcast at broadcast.jl:439 overwritten in
module ImageFiltering at /root/.julia/packages/ImageFiltering/dR6N9/src/ImageFiltering.jl:28.
WARNING: Method definition _bcs1(Any, Any) in module Broadcast at broadcast.jl:439 overwritten in
module ImageFiltering at /root/.julia/packages/ImageFiltering/dR6N9/src/ImageFiltering.jl:28.
```

```
Resolving package versions...
Installed AutoHashEquals — v0.2.0
Installed Embeddings — v0.4.1
Updating `~/julia/environments/v1.0/Project.toml`
[c5bfea45] + Embeddings v0.4.1
Updating `~/julia/environments/v1.0/Manifest.toml`
[15f4f7f2] + AutoHashEquals v0.2.0
[c5bfea45] + Embeddings v0.4.1
Precompiling project...
Precompiling Embeddings
```

```
[ Info: Precompiling Embeddings [c5bfea45-b7f1-5224-a596-15500f5db411]
└ @ Base loading.jl:1192
```

```
Resolving package versions...
Installed Requires — v0.5.2
Installed Showoff — v0.3.1
Installed RecipesBase — v0.7.0
Installed Plots — v0.28.3
Installed Contour — v0.5.1
Installed PlotUtils — v0.6.1
Installed GeometryTypes — v0.7.6
Installed PlotThemes — v1.0.0
Installed FFMPEG — v0.2.4
Installed Measures — v0.3.1
Installed GR — v0.44.0
Updating `~/julia/environments/v1.0/Project.toml`
[91a5bcdd] + Plots v0.28.3
Updating `~/julia/environments/v1.0/Manifest.toml`
[d38c429a] + Contour v0.5.1
[c87230d0] + FFMPEG v0.2.4
[28b8d3ca] + GR v0.44.0
[4d00f742] + GeometryTypes v0.7.6
[442fdcd] + Measures v0.3.1
[ccf2f8ad] + PlotThemes v1.0.0
[995b91a9] + PlotUtils v0.6.1
[91a5bcdd] + Plots v0.28.3
[3cdcf5f2] + RecipesBase v0.7.0
[ae029012] ↓ Requires v1.0.0 ⇒ v0.5.2
[992d4aef] + Showoff v0.3.1
Building GR → `~/julia/packages/GR/oiZD3/deps/build.log`
Building FFMPEG → `~/julia/packages/FFMPEG/guNlx/deps/build.log`
Building Plots → `~/julia/packages/Plots/RsO3g/deps/build.log`
Precompiling project...
Precompiling Plots
```

```
[ Info: Precompiling Plots [91a5bcdd-55d7-5caf-9e0b-520d859cae80]
└ @ Base loading.jl:1192
```

```
Resolving package versions...
Installed Adapt — v1.0.0
```

```

Installed CEnum ————— v0.2.0
Installed CUDAapi ————— v1.2.0
Installed AbstractFFTs — v0.4.1
Installed GPUArrays ——— v1.0.4
Installed FFTW ————— v1.0.1
Installed CUDAnative ——— v2.4.0
Installed CuArrays ——— v1.3.0
Installed NNlib ————— v0.6.0
Installed TimerOutputs — v0.5.3
Installed CUDAdrv ————— v3.1.0
Installed FillArrays ——— v0.7.4
Installed LLVM ————— v1.3.2
Updating `~/julia/environments/v1.0/Project.toml`
[3a865a2d] + CuArrays v1.3.0
Updating `~/julia/environments/v1.0/Manifest.toml`
[621f4979] ↓ AbstractFFTs v0.5.0 ⇒ v0.4.1
[79e6a3ab] + Adapt v1.0.0
[fa961155] + CEnum v0.2.0
[3895d2a7] + CUDAapi v1.2.0
[c5f51814] + CUDAdrv v3.1.0
[be33ccc6] + CUDAnative v2.4.0
[3a865a2d] + CuArrays v1.3.0
[7a1cc6ca] ↓ FFTW v1.1.0 ⇒ v1.0.1
[1a297f60] + FillArrays v0.7.4
[0c68f7d7] + GPUArrays v1.0.4
[929cbde3] + LLVM v1.3.2
[872c559c] + NNlib v0.6.0
[a759f4b9] + TimerOutputs v0.5.3
Building FFTW → `~/julia/packages/FFTW/MJ7kl/deps/build.log`

```

```

└ Info: Precompiling CuArrays [3a865a2d-5b23-5a0f-bc46-62713ec82fae]
└ @ Base loading.jl:1192

```

Resolving package versions...

```

Internal error: encountered unexpected error in runtime:
BoundsError{a=Array{Core.Compiler.BasicBlock, (32,)}[
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=1, last=6), preds=Array{Int64, (1,)}[32], succs=Array{Int64, (1,)}[2]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=7, last=13), preds=Array{Int64, (1,)}[1], succs=Array{Int64, (2,)}[5, 3]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=14, last=19), preds=Array{Int64, (1,)}[2], succs=Array{Int64, (1,)}[4]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=20, last=20), preds=Array{Int64, (1,)}[3], succs=Array{Int64, (1,)}[7]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=21, last=27), preds=Array{Int64, (1,)}[2], succs=Array{Int64, (1,)}[6]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=28, last=28), preds=Array{Int64, (1,)}[5], succs=Array{Int64, (1,)}[7]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=29, last=46), preds=Array{Int64, (2,)}[4, 6], succs=Array{Int64, (2,)}[9, 8]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=47, last=48), preds=Array{Int64, (1,)}[7], succs=Array{Int64, (0,)}[]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=49, last=49), preds=Array{Int64, (1,)}[7], succs=Array{Int64, (1,)}[10]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=50, last=51), preds=Array{Int64, (1,)}[9], succs=Array{Int64, (1,)}[11]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=52, last=52), preds=Array{Int64, (1,)}[10], succs=Array{Int64, (1,)}[12]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=53, last=53), preds=Array{Int64, (1,)}[11], succs=Array{Int64, (1,)}[13]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=54, last=55), preds=Array{Int64, (1,)}[12], succs=Array{Int64, (1,)}[14]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=56, last=64), preds=Array{Int64, (1,)}[13], succs=Array{Int64, (1,)}[15]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=65, last=69), preds=Array{Int64, (1,)}[14], succs=Array{Int64, (2,)}[17, 16]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=70, last=72), preds=Array{Int64, (1,)}[15], succs=Array{Int64, (0,)}[]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=73, last=80), preds=Array{Int64, (1,)}[15], succs=Array{Int64, (2,)}[19, 18]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=81, last=82), preds=Array{Int64, (1,)}[17], succs=Array{Int64, (0,)}[]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=83, last=83), preds=Array{Int64, (1,)}[17], succs=Array{Int64, (1,)}[20]),
]

```

```

Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=84, last=85), preds=Array{Int64,
(1,)}[19], succs=Array{Int64, (1,)}[21]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=86, last=86), preds=Array{Int64,
(1,)}[20], succs=Array{Int64, (1,)}[22]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=87, last=87), preds=Array{Int64,
(1,)}[21], succs=Array{Int64, (1,)}[23]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=88, last=88), preds=Array{Int64,
(1,)}[22], succs=Array{Int64, (1,)}[24]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=89, last=90), preds=Array{Int64,
(1,)}[23], succs=Array{Int64, (1,)}[25]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=91, last=91), preds=Array{Int64,
(1,)}[24], succs=Array{Int64, (1,)}[26]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=92, last=92), preds=Array{Int64,
(1,)}[25], succs=Array{Int64, (1,)}[27]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=93, last=93), preds=Array{Int64,
(1,)}[26], succs=Array{Int64, (2,)}[29, 28]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=94, last=96), preds=Array{Int64,
(1,)}[27], succs=Array{Int64, (0,)}[]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=97, last=100), preds=Array{Int64,
(1,)}[27], succs=Array{Int64, (2,)}[31, 30]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=101, last=101), preds=Array{Int64,
(1,)}[29], succs=Array{Int64, (1,)}[32]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=102, last=102), preds=Array{Int64,
(1,)}[29], succs=Array{Int64, (1,)}[32]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=103, last=105), preds=Array{Int64,
(2,)}[30, 31], succs=Array{Int64, (1,)}[1]), i=(0,))
rec_backtrace at /buildworker/worker/package_linux64/build/src/stackwalk.c:94
record_backtrace at /buildworker/worker/package_linux64/build/src/task.c:246
jl_throw at /buildworker/worker/package_linux64/build/src/task.c:577
jl_bounds_error_ints at /buildworker/worker/package_linux64/build/src/rtutils.c:187
getindex at ./array.jl:731
jfp_ptr_getindex_2271.clone_1 at /usr/local/lib/julia/sys.so (unknown line)
jl_apply_generic at /buildworker/worker/package_linux64/build/src/gf.c:2162
replace_code_newstyle! at ./compiler/ssair/legacy.jl:86
optimize at ./compiler/optimize.jl:212
typeinf at ./compiler/typeinfer.jl:35
typeinf_ext at ./compiler/typeinfer.jl:574
typeinf_ext at ./compiler/typeinfer.jl:611
jfp_ptr_typeinf_ext_1.clone_1 at /usr/local/lib/julia/sys.so (unknown line)
jl_apply_generic at /buildworker/worker/package_linux64/build/src/gf.c:2162
jl_apply at /buildworker/worker/package_linux64/build/src/julia.h:1537 [inlined]
jl_apply_with_saved_exception_state at /buildworker/worker/package_linux64/build/src/rtutils.c:257
jl_type_infer at /buildworker/worker/package_linux64/build/src/gf.c:253
jl_compile_method_internal at /buildworker/worker/package_linux64/build/src/gf.c:1764 [inlined]
jl_fptr_trampoline at /buildworker/worker/package_linux64/build/src/gf.c:1808
jl_apply_generic at /buildworker/worker/package_linux64/build/src/gf.c:2162
jl_apply at /buildworker/worker/package_linux64/build/src/julia.h:1537 [inlined]
start_task at /buildworker/worker/package_linux64/build/src/task.c:268
unknown function (ip: 0xffffffffffffffff)

```

```

Installed DiffRules ————— v0.1.0
Installed Juno ————— v0.7.2
Installed Media ————— v0.5.0
Installed DiffResults ————— v0.0.4
Installed AbstractTrees ————— v0.2.1
Installed ZipFile ————— v0.8.3
Installed CommonSubexpressions — v0.2.0
Installed Flux ————— v0.9.0
Installed CodecZlib ————— v0.6.0
Installed Tracker ————— v0.2.5
Installed ForwardDiff ————— v0.10.7
Installed TranscodingStreams — v0.9.5
Updating `~/julia/environments/v1.0/Project.toml`
[587475ba] + Flux v0.9.0
Updating `~/julia/environments/v1.0/Manifest.toml`
[1520ce14] + AbstractTrees v0.2.1
[944b1d66] + CodecZlib v0.6.0
[bbf7d656] + CommonSubexpressions v0.2.0
[163ba53b] + DiffResults v0.0.4
[b552c78f] + DiffRules v0.1.0
[587475ba] + Flux v0.9.0
[f6369f11] + ForwardDiff v0.10.7
[e5e0dc1b] + Juno v0.7.2
[e89f7d12] + Media v0.5.0
[9f7883ad] + Tracker v0.2.5
[3bb67fe8] + TranscodingStreams v0.9.5

```



```
[a5390f91] + zipfile v0.8.3
[9abbd945] + Profile
Building ZipFile → `~/julia/packages/ZipFile/oD4uG/deps/build.log`
Building CodecZlib → `~/julia/packages/CodecZlib/5t9zO/deps/build.log`
```

```
[ Info: Precompiling Flux [587475ba-b771-5e3f-ad9e-33799f191a9c]
└ @ Base loading.jl:1192
Internal error: encountered unexpected error in runtime:
BoundsError{a=Array{Core.Compiler.BasicBlock, (32,)}[
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=1, last=6), preds=Array{Int64, (1,)}[32], succs=Array{Int64, (1,)}[2]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=7, last=13), preds=Array{Int64, (1,)}[1], succs=Array{Int64, (2,)}[5, 3]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=14, last=19), preds=Array{Int64, (1,)}[2], succs=Array{Int64, (1,)}[4]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=20, last=20), preds=Array{Int64, (1,)}[3], succs=Array{Int64, (1,)}[7]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=21, last=27), preds=Array{Int64, (1,)}[2], succs=Array{Int64, (1,)}[6]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=28, last=28), preds=Array{Int64, (1,)}[5], succs=Array{Int64, (1,)}[7]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=29, last=46), preds=Array{Int64, (2,)}[4, 6], succs=Array{Int64, (2,)}[9, 8]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=47, last=48), preds=Array{Int64, (1,)}[7], succs=Array{Int64, (0,)}[]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=49, last=49), preds=Array{Int64, (1,)}[7], succs=Array{Int64, (1,)}[10]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=50, last=51), preds=Array{Int64, (1,)}[9], succs=Array{Int64, (1,)}[11]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=52, last=52), preds=Array{Int64, (1,)}[10], succs=Array{Int64, (1,)}[12]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=53, last=53), preds=Array{Int64, (1,)}[11], succs=Array{Int64, (1,)}[13]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=54, last=55), preds=Array{Int64, (1,)}[12], succs=Array{Int64, (1,)}[14]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=56, last=64), preds=Array{Int64, (1,)}[13], succs=Array{Int64, (1,)}[15]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=65, last=69), preds=Array{Int64, (1,)}[14], succs=Array{Int64, (2,)}[17, 16]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=70, last=72), preds=Array{Int64, (1,)}[15], succs=Array{Int64, (0,)}[]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=73, last=80), preds=Array{Int64, (1,)}[15], succs=Array{Int64, (2,)}[19, 18]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=81, last=82), preds=Array{Int64, (1,)}[17], succs=Array{Int64, (0,)}[]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=83, last=83), preds=Array{Int64, (1,)}[17], succs=Array{Int64, (1,)}[20]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=84, last=85), preds=Array{Int64, (1,)}[19], succs=Array{Int64, (1,)}[21]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=86, last=86), preds=Array{Int64, (1,)}[20], succs=Array{Int64, (1,)}[22]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=87, last=87), preds=Array{Int64, (1,)}[21], succs=Array{Int64, (1,)}[23]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=88, last=88), preds=Array{Int64, (1,)}[22], succs=Array{Int64, (1,)}[24]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=89, last=90), preds=Array{Int64, (1,)}[23], succs=Array{Int64, (1,)}[25]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=91, last=91), preds=Array{Int64, (1,)}[24], succs=Array{Int64, (1,)}[26]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=92, last=92), preds=Array{Int64, (1,)}[25], succs=Array{Int64, (1,)}[27]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=93, last=93), preds=Array{Int64, (1,)}[26], succs=Array{Int64, (2,)}[29, 28]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=94, last=96), preds=Array{Int64, (1,)}[27], succs=Array{Int64, (0,)}[]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=97, last=100), preds=Array{Int64, (1,)}[27], succs=Array{Int64, (2,)}[31, 30]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=101, last=101), preds=Array{Int64, (1,)}[29], succs=Array{Int64, (1,)}[32]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=102, last=102), preds=Array{Int64, (1,)}[29], succs=Array{Int64, (1,)}[32]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=103, last=105), preds=Array{Int64, (2,)}[30, 31], succs=Array{Int64, (1,)}[1]), i=(0,)}
rec_backtrace at /buildworker/worker/package_linux64/build/src/stackwalk.c:94
record_backtrace at /buildworker/worker/package_linux64/build/src/task.c:246
jl_throw at /buildworker/worker/package_linux64/build/src/task.c:577
il_bounds_error_ints at /buildworker/worker/package_linux64/build/src/rtutils.c:187
```

```

jl_bound_error_line at /buildworker/worker/package_linux64/build/src/rtutils.c:107
getindex at ./array.jl:731
jfp_ptr_getindex_2271.clone_1 at /usr/local/lib/julia/sys.so (unknown line)
jl_apply_generic at /buildworker/worker/package_linux64/build/src/gf.c:2162
replace_code_newstyle! at ./compiler/ssair/legacy.jl:86
optimize at ./compiler/optimize.jl:212
typeinf at ./compiler/typeinfer.jl:35
typeinf_ext at ./compiler/typeinfer.jl:574
typeinf_ext at ./compiler/typeinfer.jl:611
jfp_ptr_typeinf_ext_1.clone_1 at /usr/local/lib/julia/sys.so (unknown line)
jl_apply_generic at /buildworker/worker/package_linux64/build/src/gf.c:2162
jl_apply at /buildworker/worker/package_linux64/build/src/julia.h:1537 [inlined]
jl_apply_with_saved_exception_state at /buildworker/worker/package_linux64/build/src/rtutils.c:257
jl_type_infer at /buildworker/worker/package_linux64/build/src/gf.c:253
jl_compile_method_internal at /buildworker/worker/package_linux64/build/src/gf.c:1764 [inlined]
jl_fptr_trampoline at /buildworker/worker/package_linux64/build/src/gf.c:1808
jl_apply_generic at /buildworker/worker/package_linux64/build/src/gf.c:2162
jl_apply at /buildworker/worker/package_linux64/build/src/julia.h:1537 [inlined]
start_task at /buildworker/worker/package_linux64/build/src/task.c:268
unknown function (ip: 0xffffffffffffffff)

```

```

Resolving package versions...
Installed IteratorInterfaceExtensions - v1.0.0
Installed TableTraits - v1.0.0
Installed PooledArrays - v0.5.2
Installed Tables - v0.2.11
Installed CategoricalArrays - v0.7.4
Installed DataFrames - v0.20.0
Installed InvertedIndices - v1.0.0
Installed DataValueInterfaces - v1.0.0
Updating `~/julia/environments/v1.0/Project.toml`
[a93c6f00] + DataFrames v0.20.0
Updating `~/julia/environments/v1.0/Manifest.toml`
[324d7699] + CategoricalArrays v0.7.4
[a93c6f00] + DataFrames v0.20.0
[e2d170a0] + DataValueInterfaces v1.0.0
[41ab1584] + InvertedIndices v1.0.0
[82899510] + IteratorInterfaceExtensions v1.0.0
[2dfb63ee] + PooledArrays v0.5.2
[3783bdb8] + TableTraits v1.0.0
[bd369af6] + Tables v0.2.11
[9fa8497b] + Future

```

```

└ Info: Precompiling DataFrames [a93c6f00-e57d-5684-b7b6-d8193f3e46c0]
└ @ Base loading.jl:1192

```

```

Resolving package versions...
Installed ProgressBars - v0.3.2
Updating `~/julia/environments/v1.0/Project.toml`
[49802e3a] + ProgressBars v0.3.2
Updating `~/julia/environments/v1.0/Manifest.toml`
[49802e3a] + ProgressBars v0.3.2

```

```

└ Info: Precompiling ProgressBars [49802e3a-d2f1-5c88-81d8-b72133a6f568]
└ @ Base loading.jl:1192

```

Assignment 2.1: A multi-class, multi-layer perceptron

Data

Fashion-MNIST

In [0]:

```

# Download the dataset.
run(`curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.
gz -o /tmp/train-images-idx3-ubyte.gz`)
run(`curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.
gz -o /tmp/train-labels-idx1-ubyte.gz`)
run(`curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.

```

```
run(`curl -fSS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz -o /tmp/t10k-images-idx3-ubyte.gz`)
run(`curl -fSS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz -o /tmp/t10k-labels-idx1-ubyte.gz`)

nothing
```

In [0]:

```
# Then load it into RAM.
vanillaxs, vanillays = MNIST.traindata(dir="/tmp")
# There is no proper train/validation/test split provided, thus we create our own.
trainxs      = vanillaxs[:, :, 1:50_000]
trainys      = vanillays[1:50_000]
validxs      = vanillaxs[:, :, 50_001:end]
validys      = vanillays[50_001:end]
vanillaxs    = vanillaxs[1:0]
vanillays    = vanillays[1:0]
testxs, testys = MNIST.testdata(dir="/tmp")

nothing
```

In [0]:

```
# Pre-process the data
using Flux: onehotbatch

# concatenate the xs and transform ys using onehotbatch
function load(images, labels)
    n, n, d = size(images)
    xs = reshape(images, (n^2, d)) |> gpu
    ys = onehotbatch(labels, 0:9) |> gpu
    (images, labels, xs, ys)
end;
```

In [0]:

```
trainimages, trainlabels, Trainxs, Trainys = load(trainxs, trainys);
validimages, validlabels, Validxs, Validys = load(validxs, validys);
testimages, testlabels, Testxs, Testys = load(testxs, testys);
```

Utils

TrainingState Struct

In [0]:

```
# TrainingState struct to used in callbacks to record training state

mutable struct TrainingState
    train_losses::Vector{Float64}
    valid_losses::Vector{Float64}
end
TrainingState() = TrainingState([], [])

function update_loss!(state::TrainingState, loss)
    push!(state.train_losses, Tracker.data(loss(Trainxs, Trainys)))
    push!(state.valid_losses, Tracker.data(loss(Validxs, Validys)))
end;
```

Score functions

In [0]:

```
accuracy(y, y_preds) = 100*(mean(y .== y_preds));
```

Prediction functions

In [0]:

```
function max_p(distr)
    preds = Vector{Int64}()
    for i in 1:size(distr)[2]
        push!(preds, argmax(distr[:, i]))
    end
    return preds .- 1;
end;
```

Activation functions

In [0]:

```
logistic(x) = 1/(1 .+ exp.(-x));
```

1.1 Implement a multi-class, multi-layer perceptron with cross-entropy loss for the Fashion-MNIST data.

In [0]:

```
#using Flux: ADAM, Dense, Chain, softmax, params

# generic function for building arbitrarily sized mlp
# activation functions are fixed to sigmoid except for last layer's softmax

function build_mlp(input_size::Int64, output_size::Int64, hids::Vector{Int64})
    # ensure this function is not being passed nonsense
    @assert length(hids) > 0
    @assert all(hids .> 0)

    layer_list = []
    push!(layer_list, Dense(input_size, hids[1], logistic))
    for i in 2:length(hids)
        push!(layer_list, Dense(hids[i-1], hids[i], logistic))
    end
    push!(layer_list, Dense(hids[end], output_size))
    push!(layer_list, softmax)

    # unpack arguments from layer_list and chain into mlp model
    return Chain(layer_list...)
end;
```

In [0]:

```
# define a test model for sanity checking purposes
#using Base.Iterators: repeated
using Flux: crossentropy

model = build_mlp(28^2, 10, [32]) |> gpu

# define training parameters
loss(x, y) = crossentropy(model(x), y)
epochs = 10
trainingdata = [(Trainxs, Trainys)]
optimiser = ADAM()

# Train Initial Model (sanity check).
results = TrainingState()
for epoch in 1:epochs
    @time Flux.train!(loss, params(model), trainingdata, optimiser, cb=() -> update_loss!(results, loss))
end;
```

```
└ Info: Building the CUDAnative run-time library for your sm_60 device, this might take a while...
└ @ CUDAnative /root/.julia/packages/CUDAnative/Lr0yj/src/compiler/rtlib.jl:173
└ Warning: calls to Base intrinsics might be GPU incompatible
└ exception = (CUDAnative.MethodSubstitutionWarning(exp(x::T) where T<:Union{Float32, Float64} i
n Base.Math at special/exp.jl:75, exp(x::Float32) in CUDAnative at
└ (CUDAnative, Base.Math, special/exp.jl, 75, exp(x::Float32) in CUDAnative at
```

```

/root/.julia/packages/CUDAnative/Lr0yj/src/device/cuda/math.jl:99),
Base.StackTraces.StackFrame[exp at exp.jl:75, #25 at broadcast.jl:49])
└ @ CUDAnative /root/.julia/packages/CUDAnative/Lr0yj/src/compiler/irgen.jl:116
└ Warning: calls to Base intrinsics might be GPU incompatible
└ exception = (CUDAnative.MethodSubstitutionWarning{exp(x::T) where T<:Union{Float32, Float64} in
Base.Math at special/exp.jl:75, exp(x::Float32) in CUDAnative at
/root/.julia/packages/CUDAnative/Lr0yj/src/device/cuda/math.jl:99),
Base.StackTraces.StackFrame[exp at exp.jl:75, #25 at broadcast.jl:49])
└ @ CUDAnative /root/.julia/packages/CUDAnative/Lr0yj/src/compiler/irgen.jl:116

```

```

29.028511 seconds (45.70 M allocations: 2.329 GiB, 5.89% gc time)
0.010262 seconds (5.56 k allocations: 243.320 KiB)
0.010313 seconds (5.63 k allocations: 279.570 KiB)
0.009742 seconds (5.63 k allocations: 243.039 KiB)
0.010491 seconds (5.62 k allocations: 241.508 KiB)
0.011586 seconds (5.62 k allocations: 241.352 KiB)
0.009760 seconds (5.62 k allocations: 243.945 KiB)
0.010616 seconds (5.62 k allocations: 241.352 KiB)
0.010562 seconds (5.63 k allocations: 252.008 KiB)
0.010107 seconds (5.62 k allocations: 250.695 KiB)

```

1.2 Explore model variants with different layer sizes, depths, etc. to find what works well on the validation set and describe the process through which you arrive at your final model.

All code for this question is turned to markdown due to long computation times

To find a good model, we began by using heuristics to iteratively explore architectures and hyperparameters, and then fine tuned the promising hyperparameters and architectures.

1. Heuristic Architecture Experimentation

In the case of an MLP we consider the architecture parameters which can vary as: the number of hidden layers, the depths of the layers and the activation functions. At this stage we kept the default values for hyperparameters such as the learning rate, so that we could heuristically investigate architectures.

First we experimented with the number of hidden layers, h . We chose $h = 2$ and increased them until the reduction in validation set loss was insignificant in comparison to the extra computation time. When using $h \geq 4$ we did not observe an increase in performance, but instead observed extreme overfitting even at a reasonably low numbers of epochs ~ 1500 . This is probably because the network has become powerful enough to simply memorise the training data. Hence, we decided to explore the following range in the grid search: $h = \{1, 2, 3\}$.

Experimenting with the number of neurons, N_h , we performed a rough grid search on a one-layer network and we found that values beyond 518 do not typically lead to a significant decrease in loss on the validation set. This led us to consider the range of $N_h = \{16, 32, 64, 128, 256, 518\}$ in the grid search.

The three most commonly used activation functions include the logistic function, tanh and ReLu. In the experimentation stage we found that tanh and ReLu gave good results on the training set, but consistently overfitted on the validation set and suffered from vanishing gradient problems, whilst the logistic function gave the best results on the validation set, and did not have problems with vanishing gradient nor stability.

Following our intuition, we decided to use softmax in the last layer as we are performing multiclass classification and the softmax delivers output which can be interpreted as a probability distribution.

We include the body of code we used to experiment with below.

```

# Code for testing initial heuristics

# Varying these parameters:
model = Chain(
    Dense(28^2, 64, relu),
    Dense(64, 10),
    softmax,
) |> gpu

# Defining training environment
loss(x, y) = crossentropy(model(x), y)
epochs = 20

```

```

epochs      = 20
trainingdata = repeated((Trainxs, Trainys), epochs)
callback     = () -> println("${loss(Trainxs, Trainys)}\t${loss(Validxs, Validys)}")
optimiser    = ADAM()

# Train Initial Model (sanity check).
results = TrainingState()
Flux.train!(loss, params(model), trainingdata, optimiser, cb=() -> update_loss!(results, loss));

println("Accuracy: ${(accuracy(validys, max_p(model(Validxs))))}% ")

```

2. Rough Network Architecture Search

Using the above defined ranges, we ran a rough architectural grid search, considering all the possible permutations of the number of layers and the number of neurons, initially running each model only for 10 epochs. This gave us an idea for how the model will begin to train, and from this we looked at loss curves to get a better understanding of how we expected a model to perform. We then selected a subset of promising architectures to train until full convergence. We include the results below:

We obtained lowest validation loss when using $N_{\{h_1\}} = N_{\{h_2\}} = N_{\{h_3\}} = 512$ hidden layers, each using 512 neurons. This, however, is a fairly big model, which inevitably led to an increase in computational time. We also observed very good results for the model where $N_{\{h_1\}} = N_{\{h_2\}} = N_{\{h_3\}} = 128$. To check whether we might achieve similar results when using the smaller model, we trained both models to convergence and compared the accuracies.

```

# Automate the experimental process
# We try all combinations of hidden layers up to 3 (4 including output)

hidden_units = [2^n for n in 4:9]

hidden_layers = []
# 1 layers
for h in hidden_units
    push!(hidden_layers, [h, 0, 0])
end;

# 2 layers
for h1 in hidden_units
    for h2 in hidden_units
        push!(hidden_layers, [h1, h2, 0])
    end
end;

# 3 layers
for h1 in hidden_units
    for h2 in hidden_units
        for h3 in hidden_units
            push!(hidden_layers, [h1, h2, h3])
        end
    end
end;

hidden_layers = reduce(hcat, hidden_layers)';
num_params = size(hidden_layers)[1];

df_hids = DataFrame(hids1 = hidden_layers[:, 1], hids2 = hidden_layers[:, 2], hids3 = hidden_layers[:, 3], Train_Loss = zeros(num_params), Val_Loss=zeros(num_params))

for row in ProgressBar(eachrow(df_hids))

    # define model

```

```

hids = [row.hids1, row.hids2, row.hids3]
hids = filter(x -> x != 0, hids)
model = build_mlp(28^2, 10, hids) |> gpu

# define training parameters
loss(x, y) = crossentropy(model(x), y)
epochs = 150
trainingdata = [(Trainxs, Trainys)]
optimiser = ADAM()

# train
results = TrainingState()
for epoch in 1:epochs
  Flux.train!(loss, params(model), trainingdata, optimiser, cb=() -> update_loss!(results, loss));
end

row.Train_Loss = minimum(results.train_losses)
row.Val_Loss = minimum(results.valid_losses)
end;

```

```
df_hids
```

```

# Exporting the results
import Pkg; Pkg.add("CSV")
using CSV
CSV.write("q1_grid_search.csv", df_hids)

```

3. Select subset of architectures

From analysing the results of the grid search, we decided to select a varied subset of architectures to proceed further with:

- $N_{\{h_1\}} = N_{\{h_2\}} = N_{\{h_3\}} = 512$, as this model gave the lowest validation loss on the grid search.
- $N_{\{h_1\}} = N_{\{h_2\}} = N_{\{h_3\}} = 128$, which also gave a relatively small loss given its simpler architecture. - $N_{\{h_1\}} = 512, N_{\{h_2\}} = 256, N_{\{h_3\}} = 0$, as it gave the third best loss, but only uses 2 hidden layers (and is much faster to train).
- $N_{\{h_1\}} = 256, N_{\{h_2\}} = 512, N_{\{h_3\}} = 256$

```

# Define subset of models:
hids_vars = [[512, 512, 512], [512, 256, 0], [256, 512, 256], [128, 128, 128]];

```

4. Heuristic Hyperparameter Experimentation

Using the subset of models, we explored different optimisers and changing the default learning rate.

We tested out the following optimisers: ADAM (which we have used so far as default), SGD and Momentum. We trained the models to convergence (~2000 epochs) and compared validation accuracies.

We include an example experimentation set-up below.

In [0]:

```
using Flux: ADAM, SGD, Momentum
```

```

# Pick one of the pre-selected models
chosen_hids = hids_vars[4]

# Define model
chosen_hids = filter(x -> x != 0, chosen_hids)

```

```

experimental_model = build_mlp(28^2, 10, chosen_hids) |> gpu

# Define training params
loss(x, y) = crossentropy(experimental_model(x), y)

epochs      = 2000
trainingdata = [(Trainxs, Trainys)]
optimiser    = Momentum(0.01)

# train until convergence
results = TrainingState()

@time for epoch in ProgressBar(1:epochs)
    Flux.train!(loss, params(experimental_model), trainingdata, optimiser, cb=() ->
update_loss!(results, loss));

end;

println("Accuracy: $(accuracy(validys, max_p(experimental_model(Validxs))))% on
experimental model: $(chosen_hids)")

# Pick one of the pre-selected models
chosen_hids = hids_vars[1]

# Define model
chosen_hids = filter(x -> x != 0, chosen_hids)
experimental_model = build_mlp(28^2, 10, chosen_hids) |> gpu

# Define training params
loss(x, y) = crossentropy(experimental_model(x), y)

epochs      = 2000
trainingdata = [(Trainxs, Trainys)]
optimiser    = ADAM(0.0001)

# train until convergence
results = TrainingState()

@time for epoch in ProgressBar(1:epochs)
    Flux.train!(loss, params(experimental_model), trainingdata, optimiser, cb=() ->
update_loss!(results, loss));

end;

println("Accuracy: $(accuracy(validys, max_p(experimental_model(Validxs))))% on
experimental model: $(chosen_hids)")

```

Using this method, we consistently found that Adam significantly outperformed both SGD and Momentum, usually by about 10%. Further experimenting with learning rates in Adam, we found the following:

- Across all models, using learning rates of 0.01 and above causes the model to struggle to converge.
- Across the majority of our selected models, using the default learning rate of 0.001 delivered the best validation accuracy.
- Across all models, using learning rates of 0.0001 and smaller we observed a drop in validation accuracy.

5. Select best model from the subset of models

Using the best hyperparameters we found above, we then trained all of our selected models to convergence and picked the one which gave the best validation accuracy.


```

hids_vars_ = reduce(hcat, hids_vars)';
experimental_params = size(hids_vars)[1];

df_model = DataFrame(hids1 = hids_vars[:, 1], hids2 = hids_vars[:, 2], hids3 =
hids_vars[:, 3], Train_Loss = zeros(experimental_params),
Val_Loss=zeros(experimental_params), Val_Acc = zeros(experimental_params));

for row in ProgressBar(eachrow(df_model))

    # define model
    hids = [row.hids1, row.hids2, row.hids3]
    hids = filter(x -> x != 0, hids)
    model = build_mlp(28^2, 10, hids) |> gpu


    # define training parameters
    loss(x, y) = crossentropy(model(x), y)

    # train till convergence
    epochs = 1500
    trainingdata = [(Trainxs, Trainys)]
    optimiser = ADAM(0.001)

    # train
    results = TrainingState()
    for epoch in 1:epochs
        Flux.train!(loss, params(model), trainingdata, optimiser, cb=() -> update_loss!(resu
lts, loss));
    end

    row.Train_Loss = minimum(results.train_losses)
    row.Val_Loss = minimum(results.valid_losses)
    row.Val_Acc = accuracy(validys, max_p(model(Validxs)))
end;

```



```

df_model

```

We defined our best model as the one which gave the highest validation accuracy.

```

best_row_ix = argmax(df_model[:, :Val_Acc])
best_row = df_model[best_row_ix, :]
best_hids = [best_row.hids1, best_row.hids2, best_row.hids3];

```

1.3 Train your final model to convergence on the training set using an optimisation algorithm of your choice.

Training our best model, $N_{h_1} = 512$, $N_{h_2} = 256$, $N_{h_3} = 0$, to convergence.

In [0]:

```

#using Flux

# Define model
#best_hids = filter(x -> x != 0, best_hids)
best_hids = [512, 256]
final_model = build_mlp(28^2, 10, best_hids) |> gpu

```

```
# Define training params
loss(x, y) = crossentropy(final_model(x), y)

# Train till convergence
epochs      = 2000
trainingdata = [(Trainxs, Trainys)]
optimiser   = ADAM()

results = TrainingState()

@time for epoch in ProgressBar(1:epochs)
    Flux.train!(loss, params(final_model), trainingdata, optimiser, cb=() -> update_loss!(results, loss));
end;
```

100.00% |██| 2000/2000 01:44<00:00, 19.22 it/s]

104.004071 seconds (17.12 M allocations: 743.013 MiB, 1.20% gc time)

1.4 Provide a plot of the loss on the training set and validation set for each epoch of training.

In [0]:

```
plt = plot(1:epochs, results.train_losses, title="Cross Entropy Loss vs Epochs for final model", label = "training")
plot!(1:epochs, results.valid_losses, label = "validation")
xlabel!("Epoch")
ylabel!("Cross Entropy")
plot(plt)
```

Out[0]:

1.5 Provide the final accuracy on the training, validation, and test set.

In [0]:

```
println("Accuracy on training data: $(accuracy(trainys, max p(final model(Trainxs))))")
```

```
└─ Warning: Performing scalar operations on GPU arrays: This is very slow, consider disallowing the
se operations with `allowscalar(false)`
└─ @ GPUArrays /root/.julia/packages/GPUArrays/tIM15/src/indexing.jl:16
```

Accuracy on training data: 97.10600000000001

In [0]:

```
println("Accuracy on validation data: $(accuracy(validys, max p(final model(Validxs))))")
```

Accuracy on validation data: 88.8

In [0]:

```
println("Accuracy on test data: $(accuracy(testys, max p(final model(Testxs))))")
```

Accuracy on test data: 88.52

1.6 Analyse the errors of your models by constructing a confusion matrix which classes are easily “confused” by the model? Hypothesise why.

Make predictions:

In [0]:

```
predicted_probs = final_model(Validxs)
predictions = Vector{Int64}()
for i in 1:10000
    push!(predictions, argmax(predicted_probs[:, i]))
end
predictions = predictions .- 1;
```

Compute confusion matrix:

In [0]:

```
using DataFrames: names!, insertcols!

label_names = [:Tshirt, :Trousr, :Pulovr, :Dress, :Coat,
               :Sandal, :Shirt, :Sneakr, :Bag, :Anboot]

function confusion_matrix(k, gts, preds)
    n = length(gts)
    length(preds) == n || throw(DimensionMismatch("Inconsistent lengths."))
    R = zeros{Int, k, k}
    for i = 1:n
        g = gts[i]
        p = preds[i]
        R[g+1, p+1] += 1
    end
    df = DataFrame(R)
    names!(df, label_names)
    insertcols!(df, 1, (:GARMENT => label_names))
    return df
end;
```

In [0]:

```
confusion_matrix(10, validys, predictions)
```

```
└ Warning: `names!(df::AbstractDataFrame, vals::Vector{Symbol}; makeunique::Bool=false)` is deprecated, use `rename!(df, vals, makeunique=makeunique)` instead.
└ caller = confusion_matrix{::Int64, ::Array{Int64,1}, ::Array{Int64,1}} at In[20]:16
└ @ Main ./In[20]:16
```

Out[0]:

10 rows × 11 columns

	GARMENT	Tshirt	Trousr	Pulovr	Dress	Coat	Sandal	Shirt	Sneakr	Bag	Anboot
	Symbol	Int64	Int64	Int64	Int64	Int64	Int64	Int64	Int64	Int64	Int64
1	Tshirt	772	0	14	31	0	1	200	0	4	1
2	Trousr	1	965	2	12	1	0	6	0	1	0
3	Pulovr	13	0	795	8	69	0	119	0	4	0
4	Dress	15	6	3	922	25	0	46	0	4	0
5	Coat	2	2	80	34	821	1	106	0	4	0
6	Sandal	0	0	0	0	0	959	0	23	6	8
7	Shirt	46	1	37	16	38	0	821	0	11	0
8	Sneakr	0	0	1	0	0	15	0	907	0	32
9	Bag	6	0	3	1	6	2	6	4	938	2
10	Anboot	0	0	0	0	0	11	0	29	1	980

As can be seen in the above confusion matrix, the model finds similarly looking items are confused for similarly looking items. For example, t-shirts are most often confused with shirts, ankle boots are occasionally confused with sandals and sneakers, pullovers with coats, etc. This suggests our model is learning what we want it to learn, namely, the ability to visually distinguish clothing items, but is not able to learn the finer details separating different types of the same clothing (e.g. t-shirt and shirt).

(e.g. t-shirt and shirt):

In [0]:

```
# Installation cell
%%shell
if ! command -v julia 2>&1 > /dev/null
then
    wget 'https://julialang-s3.julialang.org/bin/linux/x64/1.0/julia-1.0.5-linux-x86_64.tar.gz' \
        -O /tmp/julia.tar.gz
    tar -x -f /tmp/julia.tar.gz -C /usr/local --strip-components 1
    rm /tmp/julia.tar.gz
fi
julia -e "using Pkg; pkg"add IJulia; precompile;"
```

```
--2019-12-13 10:30:17-- https://julialang-s3.julialang.org/bin/linux/x64/1.0/julia-1.0.5-linux-x86_64.tar.gz
Resolving julialang-s3.julialang.org (julialang-s3.julialang.org)... 151.101.2.49, 151.101.66.49, 151.101.130.49, ...
Connecting to julialang-s3.julialang.org (julialang-s3.julialang.org)|151.101.2.49|:443...
connected.
HTTP request sent, awaiting response... 302 gce internal redirect trigger
Location: https://storage.googleapis.com/julialang2/bin/linux/x64/1.0/julia-1.0.5-linux-x86_64.tar.gz [following]
--2019-12-13 10:30:18-- https://storage.googleapis.com/julialang2/bin/linux/x64/1.0/julia-1.0.5-linux-x86_64.tar.gz
Resolving storage.googleapis.com (storage.googleapis.com)... 74.125.133.128, 2a00:1450:400c:c09::80
Connecting to storage.googleapis.com (storage.googleapis.com)|74.125.133.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 88706549 (85M) [application/octet-stream]
Saving to: '/tmp/julia.tar.gz'
```

```
/tmp/julia.tar.gz 100%[=====>] 84.60M 102MB/s in 0.8s
```

```
2019-12-13 10:30:19 (102 MB/s) - '/tmp/julia.tar.gz' saved [88706549/88706549]
```

```
Cloning default registries into /root/.julia/registries
Cloning registry General from "https://github.com/JuliaRegistries/General.git"
Resolving package versions...
Installed Conda ----- v1.3.0
Installed ZMQ ----- v1.0.0
Installed VersionParsing --- v1.2.0
Installed IJulia ----- v1.20.2
Installed Parsers ----- v0.3.10
Installed JSON ----- v0.21.0
Installed BinaryProvider --- v0.5.8
Installed MbedTLS ----- v0.6.8
Installed SoftGlobalScope - v1.0.10
Updating `~/julia/environments/v1.0/Project.toml`
[7073ff75] + IJulia v1.20.2
Updating `~/julia/environments/v1.0/Manifest.toml`
[b99e7846] + BinaryProvider v0.5.8
[8f4d0f93] + Conda v1.3.0
[7073ff75] + IJulia v1.20.2
[682c06a0] + JSON v0.21.0
[739be429] + MbedTLS v0.6.8
[69de0a69] + Parsers v0.3.10
[b85f4697] + SoftGlobalScope v1.0.10
[81def892] + VersionParsing v1.2.0
[c2297ded] + ZMQ v1.0.0
[2a0f44e3] + Base64
[ade2ca70] + Dates
[8ba89e20] + Distributed
[7b1f6079] + FileWatching
[b77e0a4c] + InteractiveUtils
[76f85450] + LibGit2
[8f399da3] + Libdl
[37e2e46d] + LinearAlgebra
[56ddb016] + Logging
[d6f4376e] + Markdown
[a63ad114] + Mmap
[44cfe95a] + Pkg
[de0858da] + Printf
[3fa0cd96] + REPL
[0a2f2841] + Random
```

```
[9a510204] + Random
[ea8e919c] + SHA
[9e88b42a] + Serialization
[6462fe0b] + Sockets
[8dfed614] + Test
[cf7118a7] + UUIDs
[4ec0a83e] + Unicode
Building Conda → `~/.julia/packages/Conda/kLXeC/deps/build.log`
Building ZMQ → `~/.julia/packages/ZMQ/ABGOx/deps/build.log`
Building MbedTLS → `~/.julia/packages/MbedTLS/X4xar/deps/build.log`
Building IJulia → `~/.julia/packages/IJulia/F1GUo/deps/build.log`
Precompiling project...
Precompiling IJulia
```

Out[0]:

In [0]:

```
using Pkg

Pkg.add(Pkg.PackageSpec(;name="CuArrays", version=v"1.3.0"))
Pkg.add(Pkg.PackageSpec(;name="Flux", version=v"0.9.0"))

pkg"add ImageMagick"
pkg"add Images"
pkg"add Plots"
pkg"add MLDatasets"

pkg"precompile"

using Base.Iterators: partition

using CuArrays
using Flux
using Flux: @epochs, mse, throttle, onehotbatch
using Images

using LinearAlgebra
using Random
using MLDatasets
using Statistics
using Plots

Pkg.add("DataFrames")
using DataFrames

using Base.Iterators: partition

Pkg.add("ProgressBars")
using ProgressBars

Pkg.add("CSV")
using CSV

Updating registry at `~/.julia/registries/General`
Updating git-repo `https://github.com/JuliaRegistries/General.git`
Resolving package versions...
Installed Adapt _____ v1.0.0
Installed Requires _____ v0.5.2
Installed Reexport _____ v0.2.0
Installed MacroTools _____ v0.5.3
Installed FFTW _____ v1.0.1
Installed GPUArrays _____ v1.0.4
Installed DataStructures _____ v0.17.6
Installed CuArrays _____ v1.3.0
Installed CEnum _____ v0.2.0
Installed NNlib _____ v0.6.0
Installed TimerOutputs _____ v0.5.3
Installed CUDAapi _____ v1.2.0
Installed AbstractFFTs _____ v0.4.1
Installed FillArrays _____ v0.7.4
Installed CUDAdrv _____ v3.1.0
Installed OrderedCollections _____ v1.1.0
Installed LLVM _____ v1.3.2
Installed CUDAnative _____ v2.4.0
```

```

Installed CUDAActive ----- v2.4.0
Updating `~/ .julia/environments/v1.0/Project.toml`
[3a865a2d] + CuArrays v1.3.0
Updating `~/ .julia/environments/v1.0/Manifest.toml`
[621f4979] + AbstractFFTs v0.4.1
[79e6a3ab] + Adapt v1.0.0
[fa961155] + CEnum v0.2.0
[3895d2a7] + CUDAapi v1.2.0
[c5f51814] + CUDAdrv v3.1.0
[be33ccc6] + CUDAnative v2.4.0
[3a865a2d] + CuArrays v1.3.0
[864edb3b] + DataStructures v0.17.6
[7a1cc6ca] + FFTW v1.0.1
[1a297f60] + FillArrays v0.7.4
[0c68f7d7] + GPUArrays v1.0.4
[929cbde3] + LLVM v1.3.2
[1914dd2f] + MacroTools v0.5.3
[872c559c] + NNlib v0.6.0
[bac558e1] + OrderedCollections v1.1.0
[189a3867] + Reexport v0.2.0
[ae029012] + Requires v0.5.2
[a759f4b9] + TimerOutputs v0.5.3
[2f01184e] + SparseArrays
[10745b16] + Statistics
Building FFTW → `~/ .julia/packages/FFTW/MJ7kl/deps/build.log`
Resolving package versions...
Installed DiffRules ----- v0.1.0
Installed AbstractTrees ----- v0.2.1
Installed FixedPointNumbers ----- v0.6.1
Installed BinDeps ----- v1.0.0
Installed CommonSubexpressions - v0.2.0
Installed Flux ----- v0.9.0
Installed ColorTypes ----- v0.8.0
Installed NaNMath ----- v0.3.3
Installed DataAPI ----- v1.1.0
Installed URIParser ----- v0.4.0
Installed Juno ----- v0.7.2
Installed SpecialFunctions ----- v0.8.0
Installed DiffResults ----- v0.0.4
Installed ZipFile ----- v0.8.3
Installed StaticArrays ----- v0.12.1
Installed Media ----- v0.5.0
Installed Missings ----- v0.4.3
Installed SortingAlgorithms ----- v0.3.1
Installed Colors ----- v0.9.6
Installed ForwardDiff ----- v0.10.7
Installed CodecZlib ----- v0.6.0
Installed StatsBase ----- v0.32.0
Installed Compat ----- v2.2.0
Installed Tracker ----- v0.2.5
Installed TranscodingStreams ----- v0.9.5
Updating `~/ .julia/environments/v1.0/Project.toml`
[587475ba] + Flux v0.9.0
Updating `~/ .julia/environments/v1.0/Manifest.toml`
[1520ce14] + AbstractTrees v0.2.1
[9e28174c] + BinDeps v1.0.0
[944b1d66] + CodecZlib v0.6.0
[3da002f7] + ColorTypes v0.8.0
[5ae59095] + Colors v0.9.6
[bbf7d656] + CommonSubexpressions v0.2.0
[34da2185] + Compat v2.2.0
[9a962f9c] + DataAPI v1.1.0
[163ba53b] + DiffResults v0.0.4
[b552c78f] + DiffRules v0.1.0
[53c48c17] + FixedPointNumbers v0.6.1
[587475ba] + Flux v0.9.0
[f6369f11] + ForwardDiff v0.10.7
[e5e0dc1b] + Juno v0.7.2
[e89f7d12] + Media v0.5.0
[e1d29d7a] + Missings v0.4.3
[77ba4419] + NaNMath v0.3.3
[a2af1166] + SortingAlgorithms v0.3.1
[276daf66] + SpecialFunctions v0.8.0
[90137ffa] + StaticArrays v0.12.1
[2913bbd2] + StatsBase v0.32.0
[9f7883ad] + Tracker v0.2.5
[3bb67fe8] + TranscodingStreams v0.9.5
[20578b4f] + URIParser v0.4.0

```

```

[50570b45] + URIParser v0.4.0
[a5390f91] + ZipFile v0.8.3
[8bb1440f] + DelimitedFiles
[9abbd945] + Profile
[1a1011a3] + SharedArrays
Building SpecialFunctions → `~/.julia/packages/SpecialFunctions/ne2iw/deps/build.log`
Building ZipFile → `~/.julia/packages/ZipFile/oD4uG/deps/build.log`
Building CodecZlib → `~/.julia/packages/CodecZlib/5t9z0/deps/build.log`
Resolving package versions...
Installed Graphics — v1.0.0
Installed MappedArrays — v0.2.2
Installed OffsetArrays — v0.11.3
Installed PaddedViews — v0.4.2
Installed ImageMagick — v0.7.5
Installed FileIO — v1.2.0
Installed ImageCore — v0.8.6
Updating `~/.julia/environments/v1.0/Project.toml`
[6218d12a] + ImageMagick v0.7.5
Updating `~/.julia/environments/v1.0/Manifest.toml`
[5789e2e9] + FileIO v1.2.0
[a2bd30eb] + Graphics v1.0.0
[a09fc81d] + ImageCore v0.8.6
[6218d12a] + ImageMagick v0.7.5
[dbb5928d] + MappedArrays v0.2.2
[6fe1bfb0] + OffsetArrays v0.11.3
[5432bcbf] + PaddedViews v0.4.2
Building ImageMagick → `~/.julia/packages/ImageMagick/vMfoS/deps/build.log`
Resolving package versions...
Installed ComputationalResources — v0.3.0
Installed WoodburyMatrices — v0.4.1
Installed ImageMorphology — v0.2.4
Installed ImageDistances — v0.2.5
Installed ImageAxes — v0.6.1
Installed Images — v0.19.1
Installed AxisArrays — v0.3.3
Installed AxisAlgorithms — v1.0.0
Installed Rotations — v0.12.0
Installed CustomUnitRanges — v0.2.0
Installed ImageTransformations — v0.8.0
Installed CatIndices — v0.2.0
Installed Interpolations — v0.12.5
Installed Distances — v0.8.2
Installed Ratios — v0.3.1
Installed IdentityRanges — v0.3.0
Installed TiledIteration — v0.2.3
Installed ImageFiltering — v0.6.7
Installed IntervalSets — v0.3.2
Installed FFTViews — v0.3.0
Installed CoordinateTransformations — v0.5.0
Installed IndirectArrays — v0.5.0
Installed RangeArrays — v0.3.1
Installed ImageShow — v0.2.0
Installed ColorVectorSpace — v0.7.1
Installed ImageMetadata — v0.7.2
Installed SimpleTraits — v0.9.1
Installed IterTools — v1.3.0
Updating `~/.julia/environments/v1.0/Project.toml`
[916415d5] + Images v0.19.1
Updating `~/.julia/environments/v1.0/Manifest.toml`
[13072b0f] + AxisAlgorithms v1.0.0
[39de3d68] + AxisArrays v0.3.3
[aafadddc] + CatIndices v0.2.0
[c3611d14] + ColorVectorSpace v0.7.1
[ed09eef8] + ComputationalResources v0.3.0
[150eb455] + CoordinateTransformations v0.5.0
[dc8bdbbb] + CustomUnitRanges v0.2.0
[b4f34e82] + Distances v0.8.2
[4f61f5a4] + FFTViews v0.3.0
[bbac6d45] + IdentityRanges v0.3.0
[2803e5a7] + ImageAxes v0.6.1
[51556ac3] + ImageDistances v0.2.5
[6a3955dd] + ImageFiltering v0.6.7
[bc367c6b] + ImageMetadata v0.7.2
[787d08f9] + ImageMorphology v0.2.4
[4e3cecf] + ImageShow v0.2.0
[02fcd773] + ImageTransformations v0.8.0
[916415d5] + Images v0.19.1
[0b125d00] + IndirectArrays v0.5.0

```



```

[9d13fd28] + IndirectArrays v0.5.0
[a98d9a8b] + Interpolations v0.12.5
[8197267c] + IntervalSets v0.3.2
[c8e1da08] + IterTools v1.3.0
[b3c3ace0] + RangeArrays v0.3.1
[c84ed2f1] + Ratios v0.3.1
[6038ab10] + Rotations v0.12.0
[699a6c99] + SimpleTraits v0.9.1
[06e1c1a7] + TiledIteration v0.2.3
[efce3f68] + WoodburyMatrices v0.4.1
Resolving package versions...
Installed Showoff ─── v0.3.1
Installed RecipesBase ─ v0.7.0
Installed FFMPEG ─── v0.2.4
Installed Plots ─── v0.28.3
Installed Contour ─── v0.5.1
Installed GeometryTypes ─ v0.7.6
Installed PlotThemes ─ v1.0.0
Installed Measures ─── v0.3.1
Installed PlotUtils ─ v0.6.1
Installed GR ─── v0.44.0
Updating `~/julia/environments/v1.0/Project.toml`
[91a5bcd] + Plots v0.28.3
Updating `~/julia/environments/v1.0/Manifest.toml`
[d38c429a] + Contour v0.5.1
[c87230d0] + FFMPEG v0.2.4
[28b8d3ca] + GR v0.44.0
[4d00f742] + GeometryTypes v0.7.6
[442fdcd] + Measures v0.3.1
[ccf2f8ad] + PlotThemes v1.0.0
[995b91a9] + PlotUtils v0.6.1
[91a5bcd] + Plots v0.28.3
[3cdcf5f2] + RecipesBase v0.7.0
[992d4aef] + Showoff v0.3.1
Building GR ──→ `~/julia/packages/GR/oiZD3/deps/build.log`
Building FFMPEG → `~/julia/packages/FFMPEG/guNlx/deps/build.log`
Building Plots → `~/julia/packages/Plots/RsO3g/deps/build.log`
Resolving package versions...
Installed GZip ─── v0.5.1
Installed MLDatasets ─ v0.4.0
Installed DataDeps ─ v0.7.1
Installed HTTP ─── v0.8.8
Installed IniFile ─ v0.5.0
Updating `~/julia/environments/v1.0/Project.toml`
[eb30cad] + MLDatasets v0.4.0
Updating `~/julia/environments/v1.0/Manifest.toml`
[124859b0] + DataDeps v0.7.1
[92fee26a] + GZip v0.5.1
[cd3eb016] + HTTP v0.8.8
[83e8ac13] + IniFile v0.5.0
[eb30cad] + MLDatasets v0.4.0
Precompiling project...
Precompiling MLDatasets

```

```

└ Info: Precompiling MLDatasets [eb30cad-4394-5ae3-aed4-317e484a6458]
└ @ Base loading.jl:1192

```

Precompiling Images

```

└ Info: Precompiling Images [916415d5-f1e6-5110-898d-aaa5f9f070e0]
└ @ Base loading.jl:1192
└ Warning: Replacing docs for `FileIO.filename :: Tuple{Any}` in module `FileIO`
└ @ Base.Docs docs/Docs.jl:223
└ Warning: Replacing docs for `FileIO.file_extension :: Tuple{Any}` in module `FileIO`
└ @ Base.Docs docs/Docs.jl:223
WARNING: Method definition _bcs1(Any, Any) in module Broadcast at broadcast.jl:439 overwritten in
module ImageFiltering at /root/.julia/packages/ImageFiltering/dR6N9/src/ImageFiltering.jl:28.
WARNING: Method definition _bcs1(Any, Any) in module Broadcast at broadcast.jl:439 overwritten in
module ImageFiltering at /root/.julia/packages/ImageFiltering/dR6N9/src/ImageFiltering.jl:28.

```

Precompiling CuArrays

```

└ Info: Precompiling CuArrays [3a865a2d-5b23-5a0f-bc46-62713ec82fae]
└ @ Base loading.jl:1192

```

Precompiling Flux

```
└ Info: Precompiling Flux [587475ba-b771-5e3f-ad9e-33799f191a9c]
└ @ Base loading.jl:1192
Internal error: encountered unexpected error in runtime:
BoundsError{a=Array{Core.Compiler.BasicBlock, (32,)}[
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=1, last=6), preds=Array{Int64,
(1,)}[32], succs=Array{Int64, (1,)}[2]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=7, last=13), preds=Array{Int64,
(1,)}[1], succs=Array{Int64, (2,)}[5, 3]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=14, last=19), preds=Array{Int64,
(1,)}[2], succs=Array{Int64, (1,)}[4]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=20, last=20), preds=Array{Int64,
(1,)}[3], succs=Array{Int64, (1,)}[7]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=21, last=27), preds=Array{Int64,
(1,)}[2], succs=Array{Int64, (1,)}[6]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=28, last=28), preds=Array{Int64,
(1,)}[5], succs=Array{Int64, (1,)}[7]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=29, last=46), preds=Array{Int64,
(2,)}[4, 6], succs=Array{Int64, (2,)}[9, 8]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=47, last=48), preds=Array{Int64,
(1,)}[7], succs=Array{Int64, (0,)}[]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=49, last=49), preds=Array{Int64,
(1,)}[7], succs=Array{Int64, (1,)}[10]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=50, last=51), preds=Array{Int64,
(1,)}[9], succs=Array{Int64, (1,)}[11]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=52, last=52), preds=Array{Int64,
(1,)}[10], succs=Array{Int64, (1,)}[12]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=53, last=53), preds=Array{Int64,
(1,)}[11], succs=Array{Int64, (1,)}[13]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=54, last=55), preds=Array{Int64,
(1,)}[12], succs=Array{Int64, (1,)}[14]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=56, last=64), preds=Array{Int64,
(1,)}[13], succs=Array{Int64, (1,)}[15]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=65, last=69), preds=Array{Int64,
(1,)}[14], succs=Array{Int64, (2,)}[17, 16]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=70, last=72), preds=Array{Int64,
(1,)}[15], succs=Array{Int64, (0,)}[]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=73, last=80), preds=Array{Int64,
(1,)}[15], succs=Array{Int64, (2,)}[19, 18]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=81, last=82), preds=Array{Int64,
(1,)}[17], succs=Array{Int64, (0,)}[]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=83, last=83), preds=Array{Int64,
(1,)}[17], succs=Array{Int64, (1,)}[20]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=84, last=85), preds=Array{Int64,
(1,)}[19], succs=Array{Int64, (1,)}[21]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=86, last=86), preds=Array{Int64,
(1,)}[20], succs=Array{Int64, (1,)}[22]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=87, last=87), preds=Array{Int64,
(1,)}[21], succs=Array{Int64, (1,)}[23]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=88, last=88), preds=Array{Int64,
(1,)}[22], succs=Array{Int64, (1,)}[24]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=89, last=90), preds=Array{Int64,
(1,)}[23], succs=Array{Int64, (1,)}[25]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=91, last=91), preds=Array{Int64,
(1,)}[24], succs=Array{Int64, (1,)}[26]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=92, last=92), preds=Array{Int64,
(1,)}[25], succs=Array{Int64, (1,)}[27]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=93, last=93), preds=Array{Int64,
(1,)}[26], succs=Array{Int64, (2,)}[29, 28]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=94, last=96), preds=Array{Int64,
(1,)}[27], succs=Array{Int64, (0,)}[]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=97, last=100), preds=Array{Int64,
(1,)}[27], succs=Array{Int64, (2,)}[31, 30]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=101, last=101), preds=Array{Int64,
(1,)}[29], succs=Array{Int64, (1,)}[32]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=102, last=102), preds=Array{Int64,
(1,)}[29], succs=Array{Int64, (1,)}[32]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=103, last=105), preds=Array{Int64,
(2,)}[30, 31], succs=Array{Int64, (1,)}[1]}, i=(0,))
rec_backtrace at /buildworker/worker/package_linux64/build/src/stackwalk.c:94
record_backtrace at /buildworker/worker/package_linux64/build/src/task.c:246
jl_throw at /buildworker/worker/package_linux64/build/src/task.c:577
jl_bounds_error_ints at /buildworker/worker/package_linux64/build/src/rutils.c:187
getindex at ./array.jl:731
```

```

jfp_ptr_getindex_2271.clone_1 at /usr/local/lib/julia/sys.so (unknown line)
jl_apply_generic at /buildworker/worker/package_linux64/build/src/gf.c:2162
replace_code_newstyle! at ./compiler/ssair/legacy.jl:86
optimize at ./compiler/optimize.jl:212
typeinf at ./compiler/typeinfer.jl:35
typeinf_ext at ./compiler/typeinfer.jl:574
typeinf_ext at ./compiler/typeinfer.jl:611
jfp_ptr_typeinf_ext_1.clone_1 at /usr/local/lib/julia/sys.so (unknown line)
jl_apply_generic at /buildworker/worker/package_linux64/build/src/gf.c:2162
jl_apply at /buildworker/worker/package_linux64/build/src/julia.h:1537 [inlined]
jl_apply_with_saved_exception_state at /buildworker/worker/package_linux64/build/src/rtutils.c:257
jl_type_infer at /buildworker/worker/package_linux64/build/src/gf.c:253
jl_compile_method_internal at /buildworker/worker/package_linux64/build/src/gf.c:1764 [inlined]
jl_fptr_trampoline at /buildworker/worker/package_linux64/build/src/gf.c:1808
jl_apply_generic at /buildworker/worker/package_linux64/build/src/gf.c:2162
jl_apply at /buildworker/worker/package_linux64/build/src/julia.h:1537 [inlined]
start_task at /buildworker/worker/package_linux64/build/src/task.c:268
unknown function (ip: 0xffffffffffffffff)

```

Precompiling ImageMagick

```

└ Info: Precompiling ImageMagick [6218d12a-5da1-5696-b52f-db25d2ecc6d1]
└ @ Base loading.jl:1192

```

Precompiling Plots

```

└ Info: Precompiling Plots [91a5bccd-55d7-5caf-9e0b-520d859cae80]
└ @ Base loading.jl:1192
Internal error: encountered unexpected error in runtime:
BoundsError{Array{Core.Compiler.BasicBlock, (32,)}}[
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=1, last=6), preds=Array{Int64, (1,)}[32], succs=Array{Int64, (1,)}[2]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=7, last=13), preds=Array{Int64, (1,)}[1], succs=Array{Int64, (2,)}[5, 3]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=14, last=19), preds=Array{Int64, (1,)}[2], succs=Array{Int64, (1,)}[4]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=20, last=20), preds=Array{Int64, (1,)}[3], succs=Array{Int64, (1,)}[7]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=21, last=27), preds=Array{Int64, (1,)}[2], succs=Array{Int64, (1,)}[6]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=28, last=28), preds=Array{Int64, (1,)}[5], succs=Array{Int64, (1,)}[7]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=29, last=46), preds=Array{Int64, (2,)}[4, 6], succs=Array{Int64, (2,)}[9, 8]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=47, last=48), preds=Array{Int64, (1,)}[7], succs=Array{Int64, (0,)}[]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=49, last=49), preds=Array{Int64, (1,)}[7], succs=Array{Int64, (1,)}[10]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=50, last=51), preds=Array{Int64, (1,)}[9], succs=Array{Int64, (1,)}[11]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=52, last=52), preds=Array{Int64, (1,)}[10], succs=Array{Int64, (1,)}[12]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=53, last=53), preds=Array{Int64, (1,)}[11], succs=Array{Int64, (1,)}[13]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=54, last=55), preds=Array{Int64, (1,)}[12], succs=Array{Int64, (1,)}[14]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=56, last=64), preds=Array{Int64, (1,)}[13], succs=Array{Int64, (1,)}[15]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=65, last=69), preds=Array{Int64, (1,)}[14], succs=Array{Int64, (2,)}[17, 16]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=70, last=72), preds=Array{Int64, (1,)}[15], succs=Array{Int64, (0,)}[]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=73, last=80), preds=Array{Int64, (1,)}[15], succs=Array{Int64, (2,)}[19, 18]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=81, last=82), preds=Array{Int64, (1,)}[17], succs=Array{Int64, (0,)}[]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=83, last=83), preds=Array{Int64, (1,)}[17], succs=Array{Int64, (1,)}[20]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=84, last=85), preds=Array{Int64, (1,)}[19], succs=Array{Int64, (1,)}[21]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=86, last=86), preds=Array{Int64, (1,)}[20], succs=Array{Int64, (1,)}[22]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=87, last=87), preds=Array{Int64, (1,)}[21], succs=Array{Int64, (1,)}[23]),

```

```

Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=88, last=88), preds=Array{Int64,
(1,)}[22], succs=Array{Int64, (1,)}[24]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=89, last=90), preds=Array{Int64,
(1,)}[23], succs=Array{Int64, (1,)}[25]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=91, last=91), preds=Array{Int64,
(1,)}[24], succs=Array{Int64, (1,)}[26]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=92, last=92), preds=Array{Int64,
(1,)}[25], succs=Array{Int64, (1,)}[27]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=93, last=93), preds=Array{Int64,
(1,)}[26], succs=Array{Int64, (2,)}[29, 28]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=94, last=96), preds=Array{Int64,
(1,)}[27], succs=Array{Int64, (0,)}[]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=97, last=100), preds=Array{Int64,
(1,)}[27], succs=Array{Int64, (2,)}[31, 30]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=101, last=101), preds=Array{Int64,
(1,)}[29], succs=Array{Int64, (1,)}[32]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=102, last=102), preds=Array{Int64,
(1,)}[29], succs=Array{Int64, (1,)}[32]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=103, last=105), preds=Array{Int64,
(2,)}[30, 31], succs=Array{Int64, (1,)}[1]}[1]), i=(0,))
rec_backtrace at /buildworker/worker/package_linux64/build/src/stackwalk.c:94
record_backtrace at /buildworker/worker/package_linux64/build/src/task.c:246
jl_throw at /buildworker/worker/package_linux64/build/src/task.c:577
jl_bounds_error_ints at /buildworker/worker/package_linux64/build/src/rtutils.c:187
getindex at ./array.jl:731
jfp_ptr_getindex_2271.clone_1 at /usr/local/lib/julia/sys.so (unknown line)
jl_apply_generic at /buildworker/worker/package_linux64/build/src/gf.c:2162
replace_code_newstyle! at ./compiler/ssair/legacy.jl:86
optimize at ./compiler/optimize.jl:212
typeinf at ./compiler/typeinfer.jl:35
typeinf_ext at ./compiler/typeinfer.jl:574
typeinf_ext at ./compiler/typeinfer.jl:611
jfp_ptr_typeinf_ext_1.clone_1 at /usr/local/lib/julia/sys.so (unknown line)
jl_apply_generic at /buildworker/worker/package_linux64/build/src/gf.c:2162
jl_apply at /buildworker/worker/package_linux64/build/src/julia.h:1537 [inlined]
jl_apply_with_saved_exception_state at /buildworker/worker/package_linux64/build/src/rtutils.c:257
jl_type_infer at /buildworker/worker/package_linux64/build/src/gf.c:253
jl_compile_method_internal at /buildworker/worker/package_linux64/build/src/gf.c:1764 [inlined]
jl_fptr_trampoline at /buildworker/worker/package_linux64/build/src/gf.c:1808
jl_apply_generic at /buildworker/worker/package_linux64/build/src/gf.c:2162
jl_apply at /buildworker/worker/package_linux64/build/src/julia.h:1537 [inlined]
start_task at /buildworker/worker/package_linux64/build/src/task.c:268
unknown function (ip: 0xffffffffffffffff)

```

Resolving package versions...

```

Installed DataValueInterfaces ————— v1.0.0
Installed InvertedIndices ————— v1.0.0
Installed PooledArrays ————— v0.5.2
Installed Tables ————— v0.2.11
Installed CategoricalArrays ————— v0.7.4
Installed DataFrames ————— v0.20.0
Installed TableTraits ————— v1.0.0
Installed IteratorInterfaceExtensions — v1.0.0
Updating `~/ .julia/environments/v1.0/Project.toml`
[a93c6f00] + DataFrames v0.20.0
Updating `~/ .julia/environments/v1.0/Manifest.toml`
[324d7699] + CategoricalArrays v0.7.4
[a93c6f00] + DataFrames v0.20.0
[e2d170a0] + DataValueInterfaces v1.0.0
[41ab1584] + InvertedIndices v1.0.0
[82899510] + IteratorInterfaceExtensions v1.0.0
[2dfb63ee] + PooledArrays v0.5.2
[3783bdb8] + TableTraits v1.0.0
[bd369af6] + Tables v0.2.11
[9fa8497b] + Future

```

```

[ Info: Precompiling DataFrames [a93c6f00-e57d-5684-b7b6-d8193f3e46c0]
└ @ Base loading.jl:1192

```

Resolving package versions...

```

Installed ProgressBars — v0.3.2
Updating `~/ .julia/environments/v1.0/Project.toml`
[49802e3a] + ProgressBars v0.3.2
Updating `~/ .julia/environments/v1.0/Manifest.toml`
[49802e3a] + ProgressBars v0.3.2

```

```
[ Info: Precompiling ProgressBars [49802e3a-d2f1-5c88-81d8-b72133a6f568]
└ @ Base loading.jl:1192
```

```
Resolving package versions...
Installed WeakRefStrings — v0.6.1
Installed FilePathsBase — v0.7.0
Installed LazyArrays — v0.13.1
Installed CSV — v0.5.18
Updating `~/ .julia/environments/v1.0/Project.toml`
[336ed68f] + CSV v0.5.18
Updating `~/ .julia/environments/v1.0/Manifest.toml`
[336ed68f] + CSV v0.5.18
[48062228] + FilePathsBase v0.7.0
[5078a376] + LazyArrays v0.13.1
[eal0d353] + WeakRefStrings v0.6.1
```

```
[ Info: Precompiling CSV [336ed68f-0bac-5ca0-87d4-7b16caf5d00b]
└ @ Base loading.jl:1192
```

Data

Fashion-MNIST

In [0]:

```
# Download the dataset.
run(`curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.
gz -o /tmp/train-images-idx3-ubyte.gz`)
run(`curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.
gz -o /tmp/train-labels-idx1-ubyte.gz`)
run(`curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.g
z -o /tmp/t10k-images-idx3-ubyte.gz`)
run(`curl -fsS http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.g
z -o /tmp/t10k-labels-idx1-ubyte.gz`)

nothing
```

In [0]:

```
# Then load it into RAM.
vanillaxs, vanillays = MNIST.traindata(dir="/tmp")
# There is no proper train/validation/test split provided, thus we create our own.
trainxs = vanillaxs[:, :, 1:50_000]
trainys = vanillays[1:50_000]
validxs = vanillaxs[:, :, 50_001:end]
validys = vanillays[50_001:end]
vanillaxs = vanillaxs[1:0]
vanillays = vanillays[1:0]
testxs, testys = MNIST.testdata(dir="/tmp")

nothing
```

In [0]:

```
function noise(x)
    # Remove one or two quadrants of the image.
    for _ in 1:2
        xoffset = rand([1, 14])
        yoffset = rand([1, 14])
        for i in xoffset:xoffset+14, j in yoffset:yoffset+14
            x[i, j] = zero(eltype(x))
        end
    end
    x
end;
```

Utilities

Training State Struct

In [0]:

```
# TrainingState struct to used in callbacks to record training state

mutable struct TrainingState
    train_losses::Vector{Float64}
    valid_losses::Vector{Float64}
end

TrainingState() = TrainingState([], [])

function update_loss!(state::TrainingState, loss)
    push!(state.train_losses, Tracker.data(loss(xs_noised, xs)))
    push!(state.valid_losses, Tracker.data(loss(vxs_noised, vxs)))
end;
```

- Activation function (logistic)

In [0]:

```
logistic(x) = 1/(1 .+ exp.(-x));
```

2.1 Implement a denoising autoencoder with mean squared error loss for the Fashion-MNIST data.

Dataset preparation and reshaping

In [0]:

```
using Flux: onehotbatch

function load(images, labels)
    n, n, d = size(images)
    xs = reshape(images, (n^2, d)) |> gpu
    ys = onehotbatch(labels, 0:9) |> gpu
    (images, labels, xs, ys)
end

trainxs_noised = deepcopy(trainxs)
for i in 1:size(trainxs)[3]
    trainxs_noised[:, :, i] = noise(trainxs[:, :, i])
end

validxs_noised = deepcopy(validxs)
for i in 1:size(validxs)[3]
    validxs_noised[:, :, i] = noise(validxs[:, :, i])
end

testxs_noised = deepcopy(testxs)
for i in 1:size(testxs)[3]
    testxs_noised[:, :, i] = noise(testxs[:, :, i])
end

trainimages, trainlabels, Trainxs, Trainys = load(trainxs, trainys);
trainimages_noised, trainlabels, Trainxs, Trainys = load(trainxs_noised, trainys);
validimages, validlabels, Validxs, Validys = load(validxs, validys);
validimages_noised, validlabels, Validxs, Validys = load(validxs_noised, validys);
testimages, testlabels, Testxs, Testys = load(testxs, testys);
testimages_noised, testlabels, testxs, testys = load(testxs_noised, testys);
```

In [0]:

```
function make_list(images)
    images_list = []

    for i in 1:size(images)[3]
        push!(images_list, images[:, :, i])
    end
end
```

```

        push!(images_list, images[:, :, 1])
    end

    return images_list
end

trainimages_ = make_list(trainimages);
trainimages_noised_ = make_list(trainimages_noised);
validimages_ = make_list(validimages);
validimages_noised_ = make_list(validimages_noised);
testimages_ = make_list(testimages);
testimages_noised_ = make_list(testimages_noised);

xs_noised = gpu.([float(hcat(vec.(imgs)...)) for imgs in partition(trainimages_noised_, 2000)])
xs = gpu.([float(hcat(vec.(imgs)...)) for imgs in partition(trainimages_, 2000)])
vxs_noised = gpu.([float(hcat(vec.(imgs)...)) for imgs in partition(validimages_noised_, 2000)])
vxs = gpu.([float(hcat(vec.(imgs)...)) for imgs in partition(validimages_, 2000)])
txs_noised = gpu.([float(hcat(vec.(imgs)...)) for imgs in partition(testimages_noised_, 2000)])
txs = gpu.([float(hcat(vec.(imgs)...)) for imgs in partition(testimages_, 2000)]);

```

An initial implementation of a denoising autoencoder with mean squared error loss. NB: we later define a more general constructor function that allows us to more easily experiment with architectural parameters - please see below.

In [0]:

```

zsize = 64

encoder = Dense(28^2, zsize, sigmoid)
decoder = Dense(zsize, 28^2, sigmoid)

m = Chain(encoder, decoder) |> gpu
loss(x, y) = mse(m(x), y)
results = TrainingState()
function loss_(X, Y)
    avg = 0
    for i in 1:size(X)[1]
        avg += loss(X[i], Y[i])
    end
    return avg / size(X)[1]
end

opt = ADAM()

#Train initial test model
@time for epoch in 1:100
    Flux.train!(loss, params(m), zip(xs_noised,xs), opt)
    update_loss!(results, loss_)
    flush(stdout)
end

```

```

└ Info: Building the CUDAnative run-time library for your sm_61 device, this might take a while...
└ @ CUDAnative /root/.julia/packages/CUDAnative/Lr0yj/src/compiler/rtlib.jl:173

```

28.296221 seconds (47.97 M allocations: 2.327 GiB, 5.68% gc time)

In [0]:

```

min_loss = round(findmin(results.valid_losses)[1], digits=10)
print("Minimum loss on validation set using the initial model: $(min_loss)")

```

Minimum loss on validation set using the initial model: 0.0242282115

2.2 Explore model variants with different layer sizes, depths, etc. to find what works well on the validation set and describe the process through which you arrive at your final model.

Defining properties of a "good model" in this setting

We feel it is necessary to initially make clear our interpretation of this question:

Whilst it is clear that the other questions focus on classification tasks, this question is slightly different, as we are asked to create a good denoising autoencoder for the Fashion MNIST data.

The primary purpose of an autoencoder is as a dimensionality reduction technique: we wish to take input data, and map it into a latent more 'meaningful' space. Embedding our data as such can then make classification/regression/other downstream ML tasks more efficient and effective, since we ideally capture the important latent structure (i.e. the important information in an image has little to do with the individual pixel values of an image, indeed, this is a rather verbose and unhelpful representation. Rather, what is important are the semantic/topological/texture/colour etc. properties of the subject and background of the image, and this is what an autoencoder seeks to recover).

A denoising autoencoder is a variation of a vanilla autoencoder, where the denoising aspect acts as a form of regularization against simply learning the identity function, since this is not a useful latent representation.

Hence, one of the key aspects of a 'good model' here in this respect is selecting a z-size which is relatively small with respect to the input (since we're trying to recover latent structure of a few variables that describe the observed data). In an ideal scenario, the denoised input should be almost exactly recoverable from this (z-size)-dimensional embedding - if this is the case, and given that our embedding is of significantly lower dimension than our input, then we know that our embedding is "semantically dense", in the sense that it has stored sufficient latent information to represent the input.

The denoising autoencoder hence aims to train the network to ignore noise in the inputs and reconstruct the inputs as accurately as possible, using the latent variables which should be robust to the artificially introduced noise.

Hence, in this question, a 'good model' is defined as a model which balances low-dimensional z-size with denoised reconstruction error/loss, as defined by the mean squared error on the input and output of the denoised autoencoder. i.e. we cannot simply try to optimise for loss, since doing so with a larger z-size would be trivial, but would fail to achieve the objective of an autoencoder.

1. Heuristic Architecture Experimentation

Once again, our aim is not necessarily to find the "optimum" model since this would require more time and resources than we have available. We instead aim to find a reasonably good model that performs well in terms of dimensionality reduction and reconstruction error on the validation set in a reasonable amount of time.

We take the same general approach as before, by first identifying promising architectures. In this model, our architecture is defined by the architecture of our encoder and decoder (i.e. number units/sizes of hidden layers and activation functions), as well as by the z-size representing our encoded vector.

Heuristic choice for z_sizes

As previously discussed, core to finding a 'good model' in this setting is selecting a z-size of sufficiently low-dimension such that the model is forced to recover latent structure, and sufficiently high-dimension that the majority of the information present in the pixel data can be restored. We heuristically chose representation dimensions to experiment with of [16, 32, 64] - this was motivated by the idea that we had 10 classes with significant inter-class variability, so we would probably require significantly more than 10 embedded dimensions to represent any usable latent features. However, we chose to limit our z_size to 64, since our input data had dimension of 784, and pushing any higher (e.g. to 128) begins to miss the point of encoding.

Number of hidden layers

We now have two architectures to define - one for the encoder, and one for the decoder, and since our architecture search in question 1 was time-consuming on its own, we can't feasibly do large searches over two neural architectures and their possible pairings. Hence, we were mostly limited to manual experimentation. One piece of initial intuition that guided us to begin with was the idea that "encoding" should be about as difficult as "decoding", since the functions are inverses of one another, and will "work together" to define an encoding, so intuitively making one model significantly more powerful may not make sense. Our initial experiments agreed with this intuition, suggesting that a symmetrical number of hidden layers is more effective than the asymmetrical case.

An example of an autoencoder with an symmetrical number of hidden layers:

```
encoder = Chain(Dense(28^2, 64), Dense(64, 256), Dense(256, zsize, sigmoid))
decoder = Chain(Dense(zsize, 64), Dense(64, 256), Dense(256, 28^2, sigmoid))
```

All experiments that followed hence assumed a symmetric architecture, which simplified our search space significantly. (Notice that we only need to specify the architecture for the encoder in our build_autoenc function)

Number of neurons

After reading around, we initially expected that autoencoders with symmetrical neurons, such as ENC(128 -> 64) -> Z -> DEC(64 -> 128), will be more effective than asymmetrical ones, such as ENC(128 -> 64) -> Z -> DEC(128 -> 64), following an

intuition of how autoencoders process information. However, this intuition seemed to apply more heavily to autoencoders that made use of convolutional neural networks (which we did not go as far as implementing).

Hence we remained sceptical of this assumption and manually tested a few architectures for ourselves:

((Just to clarify what we mean here, here's an example of an autoencoder with a symmetric architecture)):

```
encoder = Chain(Dense(28^2, 64), Dense(64, 256), Dense(256, zsize, sigmoid))
decoder = Chain(Dense(zsize, 256), Dense(256, 64), Dense(64, 28^2, sigmoid))
```

2. Rough Architecture Search

In our initial architecture experiments, we ran low-epoch exploratory searches over each value of the z -values (contained in different dataframes), so as to identify a good architecture for each selected z_size .

Number of hidden layers

To choose a suitable range for this parameter, we perform some low-epoch exploratory searches, varying the number of hidden layers, h . We find consistently that using $h \geq 3$ for both the encoder and decoder results in negligible improvement in loss and a great increase in computational complexity. The great increase in computational complexity is to be expected, especially when we consider that we are essentially finding an architecture with twice the number of hidden layers. This further agrees with our findings from Q1, where we find that training multiple models of more than 4 hidden layers will be computationally difficult for any reasonable number of epochs. Hence, we restrict our search of a 'good model' to $h = 0, 1, 2$.

Number of neurons

Using our intuitions and architectural searches from Q1 (given that we are in a similar problem domain), we choose to vary the number of hidden layers, N_h , from the following set: $N_h = \{32, 64, 128\}$. We initially start with relatively small N_h and work our way up, so that we can stop searching once we start to see the model performance cease to increase significantly with respect to the increased computational complexity required for training.

Activation Functions

We experimented with both ReLu and Sigmoid activations:

- Relu: noticed highly stochastic behaviour in the loss curves, issues with early overfitting.
- Sigmoid: smoother loss curves, and significantly less premature overfitting hence this is our final choice.

To perform searches over parameters, we found it useful as in Q1 to make a fairly generic function to create an out-of-box autoencoder of given hidden layer sizes:

In [0]:

```
function build_autoenc(input_size::Int64, z_size::Int64, hids::Vector{Int64})
    # ensure this function is not being passed nonsense
    @assert length(hids) > 0
    @assert all(hids .> 0)

    # Encoder
    enc_layer_list = []
    push!(enc_layer_list, Dense(input_size, hids[1], sigmoid))
    for i in 2:length(hids)
        push!(enc_layer_list, Dense(hids[i-1], hids[i], sigmoid))
    end
    push!(enc_layer_list, Dense(hids[end], z_size))

    # Decoder
    dec_layer_list = []
    push!(dec_layer_list, Dense(z_size, hids[1], sigmoid))
    for i in 2:length(hids)
        push!(dec_layer_list, Dense(hids[i-1], hids[i], sigmoid))
    end
    push!(dec_layer_list, Dense(hids[end], input_size))

    # unpack arguments from layer_list and chain into mlp model
    return Chain(Chain(enc_layer_list...), Chain(dec_layer_list...))
end;
```

In [0]:

```
# sanity check

m = build_autoenc(28^2, 10, [32]) |> gpu

loss(x, y) = mse(m(x), y)
results = TrainingState()
function loss_(X, Y)
    avg = 0
    for i in 1:size(X)[1]
        avg += loss(X[i], Y[i])
    end
    return avg / size(X)[1]
end

opt = ADAM(0.01)

#Train initial test model
@time for epoch in ProgressBar(1:5)
    Flux.train!(loss, params(m), zip(xs_noised,xs), opt)
    update_loss!(results, loss_)
    flush(stdout)
end
```

Example: Automating the experimental process for hidden layers, example code for how we defined a search over up to 3LPs in encoder and decoder:

```
# We search more thoroughly over combinations of hidden layers up to 2 (3 including output)

hidden_units = [2^n for n in 3:7]

hidden_layers = []

# 1 layers
for h in hidden_units
    push!(hidden_layers, [h, 0])
end;

# 2 layers
for h1 in hidden_units
    for h2 in hidden_units
        push!(hidden_layers, [h1, h2])
    end
end;

hidden_layers = reduce(hcat, hidden_layers)';
num_params = size(hidden_layers)[1];
```

In [0]:

```

function loss_(X, Y)
    avg = 0
    for i in 1:size(X)[1]
        avg += loss(X[i], Y[i])
    end
    return avg / size(X)[1]
end

opt = ADAM()

for epoch in 1:100
    Flux.train!(loss, params(m), zip(xs_noised,xs), opt)
    update_loss!(results, loss_)
    flush(stdout)
end

row.Train_Loss = minimum(results.train_losses)
row.Val_Loss = minimum(results.valid_losses)
end

return df_hids
end;

```

```

### turned into markdown to avoid lengthy computation
z = 64
df = hids_for_z(z)
CSV.write("df_grid_${z}.csv", df)

```

Analysis of the results of architecture search for fixed `z_sizes`

We display the top four architectures for each value of `z_size` in [16, 32, 64]:

`z = 16`

hids1	hids2	train_loss	val_loss
128	0	0.02172897	0.02228609
64	0	0.02381921	0.02433414
128	128	0.02505465	0.02563540
128	64	0.02584267	0.02644741

`z = 32`

hids1	hids2	train_loss	val_loss
128	0	0.0207087	0.0212736
64	0	0.0232051	0.0237209
128	128	0.0242866	0.0249159
128	64	0.0254684	0.0260935

`z = 64`

hids1	hids2	train_loss	val_loss
128	0	0.02007155	0.02062156
64	0	0.02243861	0.02293160
128	128	0.02425469	0.02488294
128	64	0.02484051	0.02542849

Lucky for us, the top four architectures for each `z_size` all appear to be the same!

3. Select a subset of models

We choose to proceed by focusing on the top four architectures that consistently performed the best in the previous rough search. Note that we assumed a sigmoid activation function in our initial search, and when we varied this in our four top

models, we typically observed no significant change in validation accuracy, hence we decided to fix our architectures to use sigmoid throughout.

Chosen (symmetric) architectures for next steps:

- $N_{\{h_1\}} = 128$ for both Encoder and Decoder.
- $N_{\{h_1\}} = 64$ for both Encoder and Decoder.
- $N_{\{h_1\}} = 128, N_{\{h_2\}} = 128$ for the Encoder and vice versa for the Decoder.
- $N_{\{h_1\}} = 128, N_{\{h_2\}} = 64$ for the Encoder and vice versa for the Decoder.

We proceed to heuristically optimise hyperparameters for all of these models in order to select the best final model...

4. Hyperparameter Optimisation

We primarily focused on optimising the learning rate for ADAM.

Learning Rate

Basing our selection on standard values, we ran a simple line search of values: \$0.0001\$, \$0.001\$, and \$0.01\$, training each model to convergence and manually observing loss graphs to check for under or overfitting to determine a good rough range for learning rate, and a final model architecture to fine tune.

Example code and results for this are shown below.

In [0]:

```
# We search more thoroughly over combinations of hidden layers up to 2 (3 including output)

hidden_units = [2^n for n in 6:7]
learning_rates = [0.0001, 0.001, 0.01]

parameters = []
# 1 layers
for h in hidden_units
    for l in learning_rates
        push!(parameters, [h, 0, 1])
    end
end;

# 2 layers
for h1 in hidden_units
    for h2 in hidden_units
        for l in learning_rates
            push!(parameters, [h1, h2, 1])
        end
    end
end;

parameters = reduce(hcat, parameters)';
num_params = size(parameters)[1];
```

In [0]:

```
function hyperparams_for_z(z_size, parameters)

    df = DataFrame(hids1=parameters[:, 1], hids2=parameters[:, 2], Learning_Rate=parameters[:, 3],
                   Train_Loss = zeros(num_params), Val_Loss=zeros(num_params));

    for row in ProgressBar(eachrow(df))

        # define model
        hids = [Int(row.hids1), Int(row.hids2)]
        hids = filter(x -> x != 0, hids)

        m = build_autoenc(28^2, z_size, hids) |> gpu

        loss(x, y) = mse(m(x), y)
        results = TrainingState()

        function loss_(X, Y)
            avg = 0
            for i in 1:size(X)[1]
                avg += loss(X[i], Y[i])
            end
            avg / size(X)[1]
        end

        results = TrainingState(loss_, m, z_size)
```

```

        avg += loss(X[i], I[i])
    end
    return avg / size(X)[1]
end

opt = ADAM(row.Learning_Rate)

for epoch in 1:150
    Flux.train!(loss, params(m), zip(xs_noised,xs), opt)
    update_loss!(results, loss_)
    flush(stdout)
end

row.Train_Loss = minimum(results.train_losses)
row.Val_Loss = minimum(results.valid_losses)
end

return df
end;

```

```

### turned into markdown to avoid lengthy notebook runtime
z = 64
df = hyperparams_for_z(z, parameters)
CSV.write("df_hyperparams$(z).csv", df)

```

We again display the top four combinations for each value of z.

z=16

hids1	hids2	Learning_Rate	Train_Loss	Val_Loss
128	0	0.01	0.018201	0.019963
128	128	0.01	0.018632	0.020667
128	0	0.001	0.020342	0.021126
64	0	0.01	0.020207	0.021526

z=32

hids1	hids2	Learning_Rate	Train_Loss	Val_Loss
128	0	0.01	0.017333	0.018942
128	0	0.001	0.019211	0.019939
128	128	0.01	0.018232	0.020193
64	0	0.01	0.019198	0.020442

z=64

hids1	hids2	Learning_Rate	Train_Loss	Val_Loss
128	0	0.01	0.016767	0.018362
128	0	0.001	0.018457	0.019159
128	128	0.01	0.017465	0.01952
64	0	0.01	0.018582	0.019763

We see that a 2LP with 128 hidden units and a learning rate of 0.01 dominates the top scores. We now use these top scores to influence our final round of manual experimentation.

5. Selecting Best Model

With the aforementioned structured experiments, we have given ourselves an empirical starting point for what will work well with this problem. From this point, we manually experimented using slight variations on results from the above table, until we found our final model specification. Again, recall that in this question we were not necessarily looking for the model with the lowest loss, but we are trying to balance low-loss with low z-dimensionality. We used heuristics here such as looking at the ratio between the z_size and the denoised reconstruction loss (mse), as well as by visually evaluating images in the training set.

in the training set.

Our final model took the form:

$z = 32$, $\text{hids} = [128]$ i.e. a 128 hidden unit 2LP for both the encoder and the decoder, with a learning rate of 0.0095.

We elected to select $z = 32$ along with this model as the difference in denoised reconstruction loss for this model versus every other model we tried with $z = 64$ was very minor. However, since the loss in this context is a somewhat difficult to interpret quantity, we also looked over the results visually for many examples, and concluded that the image quality was, as far as we could tell, very similar - with both our best $z = 32$ and $z = 64$ embeddings seemingly picking up on non-complex textures and colour schemes for clothing, but struggling to reconstruct more complex patterns. $z = 16$ suffered a significant downgrade in output image quality, as the images it output tended to be excessively blurry (despite the still minimal difference in loss - which just goes to show how important these visual checks were).

We feel this choice of model satisfies the goal we set out to achieve of balancing representation size with denoised reconstruction loss.

2.3 Train your final model to convergence on the training set using an optimisation algorithm of your choice.

In [0]:

```
# Training final model to convergence

zsize = 32
epochs = 300

m = build_autoenc(28^2, zsize, [128]) |> gpu
loss(x, y) = mse(m(x), y)
results = TrainingState()

opt = ADAM(0.0095)

for epoch in ProgressBar(1:epochs)
    Flux.train!(loss, params(m), zip(xs_noised, xs), opt)
    update_loss!(results, loss_)
    flush(stdout)
end
```

UndefVarError: build_autoenc not defined

Stacktrace:

[1] top-level scope at In[5]:4

2.4 Provide a plot of the loss on the training set and validation set for each epoch of training.

In [0]:

```
plot(results.train_losses, title="MSE Loss vs Epochs", label = "training")
plot!(results.valid_losses, label = "validation")
xlabel!("Epoch")
ylabel!("Mean square loss")
```

Out[0]:

In [0]:

```
min_loss = round(findmin(results.valid_losses)[1], digits=10)
print("Minimum loss on validation set: $(min_loss)")
```

Minimum loss on validation set: 0.0185488909

2.5 Provide a selection of 32 images in their original, "noisy", and "denoised" form from the test set and comment on what the model has been able to learn and what it appears to find

challenging.

In [0]:

```
img(x::Vector) = Gray.(reshape(clamp.(x, 0, 1), 28, 28))

function reconstruct(imgs, ori)
    random = rand(1:length(ori), 32)
    origin = [ori[i] for i in random]
    before = [imgs[i] for i in random]
    after = img.(map(x -> cpu(m) (float(vec(x))).data, before))
    hcat(vcat.(origin, before, after)...)
end

nothing
```

In [0]:

```
x = reconstruct(testimages_noised_, testimages_)
Gray.(x)
```

Out[0]:



The model is typically very effective at recovering the shape of the original image, as well as the general class of the item. It is also reasonably effective at recovering shades and colouring. Overall, the model reconstructs the structure of the items in the image very well, though if there is too much of an item obscured by noise, it can return uncertain, blurry images (though still with similar structure to the original). The model does a satisfactory job at picking up apparent textures, though we noticed that it appears to struggle with what we believe to be leather clothing items, such as boots, jackets, and handbags (these appear significantly more blurry than other item types). The model is not able to recover more intricate details present in the image, such as logos on shoes or t-shirts - to do this would likely require a far more powerful approach to autoencoding, and perhaps more data.

In [0]:

```
# Installation cell
%%shell
if ! command -v julia 2>&1 > /dev/null
then
    wget 'https://julialang-s3.julialang.org/bin/linux/x64/1.0/julia-1.0.5-linux-x86_64.tar.gz' \
        -O /tmp/julia.tar.gz
    tar -x -f /tmp/julia.tar.gz -C /usr/local --strip-components 1
    rm /tmp/julia.tar.gz
fi
julia -e "using Pkg; pkg\"add IJulia; precompile;\""
```

Unrecognized magic `%%shell`.

Julia does not use the IPython `%magic` syntax. To interact with the IJulia kernel, use `IJulia.somefunction(...)`, for example. Julia macros, string macros, and functions can be used to accomplish most of the other functionalities of IPython magics.

In [0]:

```
using Pkg

Pkg.add(Pkg.PackageSpec(;name="CuArrays", version=v"1.3.0"))
Pkg.add(Pkg.PackageSpec(;name="Flux", version=v"0.9.0"))

pkg"add Plots"
pkg"add StatsBase"
Pkg.add("ProgressMeter")

pkg"precompile"

using CuArrays
using Flux
using Flux: onehot, chunk, batchseq, throttle, crossentropy, onecold
using StatsBase: wsample

using Base.Iterators: partition
using LinearAlgebra
using Random
using Statistics
using ProgressMeter
```

```
Resolving package versions...
Updating `~/julia/environments/v1.0/Project.toml`
[no changes]
Updating `~/julia/environments/v1.0/Manifest.toml`
[no changes]
Resolving package versions...
Updating `~/julia/environments/v1.0/Project.toml`
[no changes]
Updating `~/julia/environments/v1.0/Manifest.toml`
[no changes]
Resolving package versions...
Updating `~/julia/environments/v1.0/Project.toml`
[no changes]
Updating `~/julia/environments/v1.0/Manifest.toml`
[no changes]
Resolving package versions...
Updating `~/julia/environments/v1.0/Project.toml`
[no changes]
Updating `~/julia/environments/v1.0/Manifest.toml`
[no changes]
Resolving package versions...
Updating `~/julia/environments/v1.0/Project.toml`
[no changes]
Updating `~/julia/environments/v1.0/Manifest.toml`
[no changes]
Precompiling project...
```


Assignment 2: “Half-bag”

2.4: Bag of vectors

Data

Stanford Sentiment Treebank

In [0]:

```
run(`curl -fsS https://nlp.stanford.edu/sentiment/trainDevTestTrees_PTB.zip -o
/tmp/trainDevTestTrees_PTB.zip`)
run(`unzip -q -o -d /tmp /tmp/trainDevTestTrees_PTB.zip`)
run(`rm -f /tmp/trainDevTestTrees_PTB.zip`)

nothing
```

In [0]:

```
function loadsst(path)
  xs = Array{String}[]
  ys = Int[]
  open(path) do file
    # Quick, dirty, and improper S-expression parsing.
    for line in eachline(file)
      soup = split(line)
      push!(ys, parse{Int, lstrip(first(soup), '(')})
      tokens = String[]
      for chunk in soup[2:end]
        endswith(chunk, ")") || continue
        push!(tokens, rstrip(chunk, ')'))
      end
      push!(xs, tokens)
    end
  end
  xs, ys
end

ssttrainxs, ssttrainys = loadsst("/tmp/trees/train.txt")
sstvalidxs, sstvalidys = loadsst("/tmp/trees/dev.txt")
ssttestxs, ssttestys = loadsst("/tmp/trees/test.txt")

nothing
```

1. Implement a character-level recurrent neural network model with a cross-entropy loss for the textual portion of the Stanford Sentiment Treebank data, where the model is to predict the next character based on the character previously observed for a given sentence.

Functions to process the data

In [0]:

```
function decompose_text(some_strings)
  all_the_chars = []
  for line in some_strings
    append!(all_the_chars, "\n") # add a symbol for a new line.
    for word in line
      for letter in split(word, "")
        append!(all_the_chars, letter)
      end
      append!(all_the_chars, " ") # add the space at the end of the word
    end
  end
  all_the_chars = convert{Array{Char,1}, all_the_chars}
```

```
return all_the_chars
end
```

Out[0]:

decompose_text (generic function with 1 method)

In [0]:

```
function charify(string_seq)
    char_seq = []
    for s in string_seq
        for c in s
            push!(char_seq, c)
        end
        push!(char_seq, ' ')
    end
    return char_seq
end;
```

In [0]:

```
function decompose_text_pad(some_strings)
    all_the_chars = []
    for line in some_strings
        line_length = []
        for word in line
            for letter in split(word, "")
                append!(all_the_chars, letter)
                append!(line_length, letter)
            end
            append!(all_the_chars, " ") # add the space at the end of the word
            append!(line_length, " ")
        end
        if size(line_length)[1] < 400
            diff = 400 - size(line_length)[1]
            for i in 1:diff
                append!(all_the_chars, "_")
            end
        end
    end
    all_the_chars = convert(Array{Char,1}, all_the_chars)
    return all_the_chars
end
```

Out[0]:

decompose_text_pad (generic function with 1 method)

In [0]:

```
### PADDING ###

trainxs_char = map(charify, ssttrainxs)
trainys_char = map(charify, ssttrainxs)
validxs_char = map(charify, sstvalidxs)
validys_char = map(charify, sstvalidxs)
testxs_char = map(charify, ssttestxs)
testys_char = map(charify, ssttestxs)

mean_seq_length = round(mean(map(length, trainxs_char)))
seq_length = maximum(map(length, trainxs_char))
pad_token = '_'

for sentence in trainxs_char
    while length(sentence) != mean_seq_length
        if length(sentence) > mean_seq_length
            pop!(sentence)
        else
            push!(sentence, pad_token)
        end
    end
end;
```

```

for sentence in trainys_char
  deleteat!(sentence, 1)
  while length(sentence) != mean_seq_length
    if length(sentence) > mean_seq_length
      pop!(sentence)
    else
      push!(sentence, pad_token)
    end
  end
end;

for sentence in validxs_char
  while length(sentence) != mean_seq_length
    if length(sentence) > mean_seq_length
      pop!(sentence)
    else
      push!(sentence, pad_token)
    end
  end
end;

for sentence in validys_char
  deleteat!(sentence, 1)
  while length(sentence) != mean_seq_length
    if length(sentence) > mean_seq_length
      pop!(sentence)
    else
      push!(sentence, pad_token)
    end
  end
end;

for sentence in testxs_char
  while length(sentence) != mean_seq_length
    if length(sentence) > mean_seq_length
      pop!(sentence)
    else
      push!(sentence, pad_token)
    end
  end
end;

for sentence in testys_char
  deleteat!(sentence, 1)
  while length(sentence) != mean_seq_length
    if length(sentence) > mean_seq_length
      pop!(sentence)
    else
      push!(sentence, pad_token)
    end
  end
end;

```

In [0]:

```

# Combine all the alphabetical characters from the 3 datasets
all_chars = Array{Char,1}
all_chars = vcat(trainxs_char, validxs_char, testxs_char)
alphabet = unique(collect(Iterators.flatten(all_chars)))

### FORMAT DATASETS AS SEQUENTIAL BATCHES ###
enc_trainxs_char = [map(c -> onehot(c, alphabet), cs) for cs in trainxs_char]
enc_validxs_char = [map(c -> onehot(c, alphabet), cs) for cs in validxs_char]
enc_testxs_char = [map(c -> onehot(c, alphabet), cs) for cs in testxs_char]
enc_trainys_char = [map(c -> onehot(c, alphabet), cs) for cs in trainys_char]
enc_validys_char = [map(c -> onehot(c, alphabet), cs) for cs in validys_char]
enc_testys_char = [map(c -> onehot(c, alphabet), cs) for cs in testys_char]

trainxs_sequence_batches = collect(partition(enc_trainxs_char, 256))
trainxs_batch_sequences = map(batchseq, trainxs_sequence_batches)
trainys_sequence_batches = collect(partition(enc_trainys_char, 256))
trainys_batch_sequences = map(batchseq, trainys_sequence_batches)

validxs_sequence_batches = collect(partition(enc_validxs_char, 256))

```

```

validxs_batch_sequences = map(batchseq, validxs_sequence_batches)
validys_sequence_batches = collect(partition(enc_validys_char, 256))
validys_batch_sequences = map(batchseq, validys_sequence_batches)

testxs_sequence_batches = collect(partition(enc_testxs_char, 256))
testxs_batch_sequences = map(batchseq, testxs_sequence_batches)
testys_sequence_batches = collect(partition(enc_testys_char, 256))
testys_batch_sequences = map(batchseq, testys_sequence_batches)

nothing

```

Accuracy function:

In [0]:

```

function accuracy(xs, ys)
  mod = model.(xs[1])
  true_val = onecold.(ys[1])
  sums = 0
  for i in 1:size(xs[1])[1]
    mod1 = mod[i]
    true_val1 = true_val[i]
    for j in 1:size(xs[1])[1]
      sums += argmax(mod1[:, j]) == true_val1[j]
    end
  end
  sums = (sums / (size(xs[1])[1])^2)*100
end

```

Out[0]:

accuracy (generic function with 1 method)

Initial model: We test our initial model to check that it trains.

In [0]:

```

# Get the dimension of onehot encoding
#N = length(unique(alphabet))
logistic(x) = 1/(1 .+ exp.(-x));
N = 95
nothing

model = Chain(
  GRU(N, 16),
  Dense(16, N),
  softmax) |> gpu

function loss(xs, ys)
  l = mean(crossentropy.(model.(gpu.(xs)), gpu.(ys)))
  Flux.reset!(model)
  l
end

optimiser = ADAM()

nothing

e = Int64[]
loss_on_train = []
loss_on_valid = []

@showprogress for epoch in 1:3
  @time Flux.train!(loss, params(model), zip(trainxs_batch_sequences, trainys_batch_sequences), optimiser)
  println("\$(epoch)\t$(loss(trainxs_batch_sequences[1], trainys_batch_sequences[1]))")
  push!(e, epoch)
  @time l_train = sum(Tracker.data.(loss.(trainxs_batch_sequences, trainys_batch_sequences)))/size(trainxs_batch_sequences)[1];
  @time l_valid = sum(Tracker.data.(loss.(validxs_batch_sequences, validys_batch_sequences)))/size(validxs_batch_sequences)[1];
  push!(loss_on_train, l_train)
  push!(loss_on_valid, l_valid)
end

```

end

```
11.740482 seconds (28.48 M allocations: 1.329 GiB, 5.52% gc time)
1 4.2830563f0 (tracked)
1.785277 seconds (3.00 M allocations: 144.318 MiB, 6.28% gc time)
```

Progress: 33% |  | ETA: 0:00:28

```
0.252855 seconds (414.60 k allocations: 22.062 MiB, 3.97% gc time)
5.261333 seconds (11.57 M allocations: 531.867 MiB, 4.35% gc time)
2 3.6782773f0 (tracked)
1.643216 seconds (2.89 M allocations: 139.893 MiB, 3.68% gc time)
```

Progress: 67% |  | ETA: 0:00:11

```
0.214635 seconds (377.52 k allocations: 20.392 MiB)
5.396898 seconds (11.61 M allocations: 534.114 MiB, 4.56% gc time)
3 3.2265208f0 (tracked)
1.611125 seconds (2.89 M allocations: 139.841 MiB, 3.74% gc time)
```

Progress: 100% |  | Time: 0:00:28

```
0.208767 seconds (377.50 k allocations: 20.326 MiB)
```

2. Explore model variants with different recurrent units (“vanilla”, LSTM, and GRU), number of layers, different depths and layer sizes for the multi-layer perceptron, etc. to find what works well on the validation set and describe the process through which you arrive at your final model.

The training time for the model is significantly larger than before, leading to more pronounced limitations in our experimentation with model variants. In contrast to previous questions, we primarily stuck to manual experimentation driven by heuristic and intuition. Though we did run "targeted" (dense, small range), and "exploratory" (sparse, large range), line and grid searches where appropriate/sensible, typically in order to challenge our intuition.

In this question, we found one of the most important variations we could make to our model did not involve the neural architecture at all, rather, we begin by discussing our model's "preprocessing" step for the sentence data.

Sequential Batches (Data Input Structure)

To understand our motivations, it is probably best to make clear our interpretation of this modelling problem, which is as follows:

We are given a dataset of sentences corresponding to movie reviews, and we would like to create a model that learns to predict the next characters in a given sentence that comes from this distribution. This means that it is important to consider each sentence as an i.i.d sample. It was somewhat easier to initially get the model working with chunked together text, as we were able to simply lift and shift the code from the character sequencer for a shakespeare script in lectures. However, this did not match our problem domain, since concatenating and mixing movie reviews violates our i.i.d. assumption and perturbs the target distribution. It would chop together movie review sentences in a way that would very likely hinder our model's learning.

Instead, we treated each sentence as a datapoint, and forced them to be the same size so that we could make use of batching to train our models more efficiently. This turned out to be a little more complicated with flux than we first anticipated! Batches consist of a collection of matrices, the first matrix in a batch corresponds to the first character of sentences in that batch, the second matrix corresponds to the second character of all the sentences in that batch, etc. We then fed batches in a sequential manner, correctly training the RNN with respect to its hidden state.

The main problem this introduced that we did not experience with the 'text chunking' method, is that we must make our sequences all the same size (so that we can group them into batches. We do this via padding (which constitutes the majority of our preprocessing step!):

Padding

We experimented with both pre-padding (adding a fixed character to the beginning of the sentences in order to fix them to the required length) and post-padding (adding a fixed character to the end of the sentences). A significant amount of the literature we found seemed to err on the side of pre-padding, though we appeared to, on average, achieve better results using post-padding.

We initially began by padding sentences to be as long as the longest sentence in our dataset. This had the advantage of maintaining as much of our data as possible, but the disadvantage being that we essentially triple the size of the data that the model must iterate over in an epoch (and not in a good way... really we're just uselessly adding noise). The model was able to learn in this manner, but, as expected, training times per epoch were longer, and time until suitable convergence was significantly longer too. The second method we used was mean-padding, i.e. we force all sentences to be the same size as the mean sentence, padding and truncating as necessary. This, by its nature, kept our data the same size, but reduced the amount of "useful" data given to our model.

Varying between these combinations we observed that on average, using the mean length of sentences led to better and more stable results, while training required less computation time than using the maximum length across all sentences. We also observed better performance using post-padding: the accuracy on the training set was significantly better on the training set using pre-padding, but the accuracies on the validation and test sets were significantly better in the case of post-padding. Based on the above, we ended up processing our data by applying post-padding based on the mean average length of all the sentences. Code for padding can be seen in our above implementation under the `### PADDING ###` comments.

Reccurent Units

Vanilla

We continued our exploration by experimenting with different recurrent units. First, we experimented with the Vanilla RNN which (has no gates). Hence, as we would expect due to its simplicity and the complexity of our data set, the Vanilla RNN performed significantly worse than the LSTM and the GRU units, which achieved both smaller losses and greater accuracies. Namely, using the Vanilla RNN, we did not observe a model which was able to achieve an accuracy of more than 37% on the validation set.

GRU, LSTM

The key difference between a GRU and an LSTM is that a GRU has two gates (reset and update gates) whereas an LSTM has three gates (namely the input, output and forget gates), so the GRU controls the flow of information like the LSTM unit, but without having to use a (long-term) memory unit. It hence exposes the full hidden content. Since we are tackling a language task involving single sentences at the character level, it seemed intuitive that the GRU unit would perform better, since our task is very short-term (i.e. predict the next character... this is far more dependent on the previous character than whatever the character was 15 characters ago). GRU also outputs information from the entire hidden unit (i.e. there's less of a need for an idea for context and long-term dependencies given our dataset's distribution being short movie reviews, and our prediction). After trying both units with different architectures we found that GRUs train significantly faster and typically perform better than the LSTMs we trained, thus agreeing with our intuition.

Assumptions about relationship with MLP architecture

We considered the architecture of the MLP and the architecture of the RNN to be somewhat separate, since we can consider the entire network as a composite function of both. However, they do interact in terms of the sequential "time/order-aware" representation that the RNN "delivers" to the MLP.

In order to reduce the effort required to manually search for a good model, we made the heuristic assumption that this representation would be reasonably independent of the final MLP classification layers at the end of the network in the following sense: by the nature of backpropagation of reduction in loss, an RNN enables the MLP to find a time-aware representation for prediction of the next character. Hence, the responsibility of finding a good representation actually belongs to the MLP, and the RNN is a "tool" it can use to do so in a time-aware fashion*. In this sense, we aimed to make our RNN sufficiently "powerful/complex", such that the if the RNN architecture can find a "good" representation for one "good" MLP, it should be able to find a "good" representation for another MLP. Once we identified a good architecture for our RNN** that seemed to at the very least not hinder the performance of any reasonable MLP we gave it, and we were able to search over the MLP architecture independently.

-* This should absolutely be taken with a massive grain of salt - we are simply trying to make the problem more tractable by making a large, and perhaps mostly unfounded, intuitive step.

The architecture that allowed us to try different MLP variants somewhat independently turned out to consist of two recurrent units, and this was verified on multiple "good" MLPs. We hypothesise that this is due to the time-dependent dimensionality reduction that two recurrent units were able to achieve before passing on to the classification layers.

MLP Architecture and Hyperparams

Architecture

Finally, we explored the architecture of the MLP, in an independent manner, as defined above. Following a process very similar to in question 1, we began by experimenting with the number of hidden layers, h . We chose $h = 1$ and increased them until the reduction in validation set loss was insignificant in comparison to the extra computation time (this was done via grid search). When using $h \geq 3$ we did not observe a significant increase in performance to compensate for the increase in computational time. We hence explored results varying the number of layers in our MLP from 1 to 2 and the number of neurons from 16 to 256. As mentioned in the previous section, we also experimented with one vs two recurrent units. We observed that two recurrent units and two MLP

hidden layers gave the best results in a reasonable timeframe. We observed that a large number of neurons (around 256) in the hidden layers of the MLP did not result in significant reduction in overall loss, and hence we chose 128 neurons in the first hidden layer, 64 neurons in the second hidden layer and a size of 128 in the recurrent units. We concluded that this model constituted a sensible trade-off of computational time and accuracy given the slow training of the model in our given data.

Activation Functions We further experimented with different activation functions (Logistic, Relu and Tanh) and observed that the logistic activation function in the hidden layer of the MLP gives the best results across different architectures. In addition, we trained our model with different optimisers (ADAM, Momentum and and SGD) and observed that ADAM results in significantly better results in the validation set.

Hyperparameters

In terms of hyperparameters, we noticed that modifying ADAM's momentum parameters typically lead to wildly stochastic behaviour during training. We tried multiple learning rates and found that the learning rate 0.01 resulted in our best results, i.e. lowest validation loss and highest validation accuracy.

Batch Size

Throughout, we adjusted the batch size to find a resonable number which gave good results while keeping the training time sufficiently short. We tried batch sizes in the range [64, 128, 256, 512] and concluded that a batch size of 256 makes training efficient while leading to satisfactory results on the validation set. This seemed to be independent of architecture.

3. Train your final model to convergence on the training set using an optimisation algorithm of your choice.

In [0]:

```
logistic(x) = 1/(1 .+ exp.(-x));
N = 95
nothing

model = Chain(
    GRU(N, 128),
    GRU(128, 128),
    Dense(128, 64, logistic),
    Dense(64, N),
    softmax) |> gpu

function loss(xs, ys)
    l = mean(crossentropy.(model.(gpu.(xs)), gpu.(ys)))
    Flux.reset!(model)
    l
end

optimiser = ADAM(0.01)

nothing

e = Int64[]
loss_on_train = []
loss_on_valid = []

#Flux.train!(loss, params(model), zip(trainxs_batch_sequences, trainys_batch_sequences),
optimiser)

@showprogress for epoch in 1:90
    @time Flux.train!(loss, params(model), zip(trainxs_batch_sequences, trainys_batch_sequences), opt
imiser)
    println("\$(epoch)\t$(loss(trainxs_batch_sequences[1], trainys_batch_sequences[1]))")
    push!(e, epoch)
    l_train =
sum(Tracker.data.(loss.(trainxs_batch_sequences,trainys_batch_sequences)))/size(trainxs_batch_seque
nces)[1];
    l_valid =
sum(Tracker.data.(loss.(validxs_batch_sequences,validys_batch_sequences)))/size(validxs_batch_seque
nces)[1];
    push!(loss_on_train, l_train)
    push!(loss_on_valid, l_valid)
end

9.346140 seconds (19.22 M allocations: 869.303 MiB, 4.33% gc time)
1 2.5351348f0 (tracked)
```

Progress: 1%|  | ETA: 0:18:28

9.143353 seconds (18.46 M allocations: 832.235 MiB, 4.32% gc time)
2 2.2285936f0 (tracked)

Progress: 2%|  | ETA: 0:18:09

9.251116 seconds (18.46 M allocations: 831.539 MiB, 4.30% gc time)
3 1.9843565f0 (tracked)

Progress: 3%|  | ETA: 0:17:57

9.329880 seconds (18.45 M allocations: 831.764 MiB, 4.41% gc time)
4 1.8371565f0 (tracked)

Progress: 4%|  | ETA: 0:17:49

9.678457 seconds (18.46 M allocations: 831.489 MiB, 6.51% gc time)
5 1.7173076f0 (tracked)

Progress: 6%|  | ETA: 0:17:44

9.150228 seconds (18.45 M allocations: 831.962 MiB, 4.24% gc time)
6 1.6072534f0 (tracked)

Progress: 7%|  | ETA: 0:17:27

8.968949 seconds (18.46 M allocations: 831.514 MiB, 4.12% gc time)
7 1.5184644f0 (tracked)

Progress: 8%|  | ETA: 0:17:09

8.903502 seconds (18.45 M allocations: 831.945 MiB, 4.12% gc time)
8 1.4432967f0 (tracked)

Progress: 9%|  | ETA: 0:16:51

8.891353 seconds (18.46 M allocations: 831.496 MiB, 4.10% gc time)
9 1.3803871f0 (tracked)

Progress: 10%|  | ETA: 0:16:35

8.828154 seconds (18.45 M allocations: 831.966 MiB, 4.09% gc time)
10 1.3264141f0 (tracked)

Progress: 11%|  | ETA: 0:16:21

8.795674 seconds (18.46 M allocations: 831.571 MiB, 4.14% gc time)
11 1.2921132f0 (tracked)

Progress: 12%|  | ETA: 0:16:05

8.845951 seconds (18.45 M allocations: 831.922 MiB, 4.09% gc time)
12 1.2665216f0 (tracked)

Progress: 13%|  | ETA: 0:15:50

8.922180 seconds (18.46 M allocations: 831.512 MiB, 4.15% gc time)
13 1.2345282f0 (tracked)

Progress: 14%|  | ETA: 0:15:27

Progress: 14%|██████| ETA: 0:15:37

8.816652 seconds (18.45 M allocations: 831.958 MiB, 4.09% gc time)
14 1.217249f0 (tracked)

Progress: 16%|██████| ETA: 0:15:23

8.770081 seconds (18.45 M allocations: 831.634 MiB, 4.13% gc time)
15 1.1902492f0 (tracked)

Progress: 17%|██████| ETA: 0:15:08

9.022489 seconds (18.45 M allocations: 831.761 MiB, 6.33% gc time)
16 1.1860919f0 (tracked)

Progress: 18%|██████| ETA: 0:14:56

8.767168 seconds (18.45 M allocations: 831.642 MiB, 4.14% gc time)
17 1.164632f0 (tracked)

Progress: 19%|██████| ETA: 0:14:42

8.773018 seconds (18.45 M allocations: 831.749 MiB, 4.15% gc time)
18 1.155885f0 (tracked)

Progress: 20%|██████| ETA: 0:14:29

8.827710 seconds (18.45 M allocations: 831.658 MiB, 4.11% gc time)
19 1.1420308f0 (tracked)

Progress: 21%|██████| ETA: 0:14:16

8.910049 seconds (18.45 M allocations: 831.751 MiB, 4.10% gc time)
20 1.1388562f0 (tracked)

Progress: 22%|██████| ETA: 0:14:03

8.756559 seconds (18.45 M allocations: 831.636 MiB, 4.07% gc time)
21 1.1283729f0 (tracked)

Progress: 23%|██████| ETA: 0:13:50

9.019294 seconds (18.45 M allocations: 831.781 MiB, 6.16% gc time)
22 1.1208802f0 (tracked)

Progress: 24%|██████| ETA: 0:13:38

8.998074 seconds (18.45 M allocations: 831.636 MiB, 4.09% gc time)
23 1.1161122f0 (tracked)

Progress: 26%|██████| ETA: 0:13:26

8.872617 seconds (18.45 M allocations: 831.747 MiB, 4.13% gc time)
24 1.1030908f0 (tracked)

Progress: 27%|██████| ETA: 0:13:13

8.799184 seconds (18.45 M allocations: 831.653 MiB, 4.10% gc time)
25 1.0980754f0 (tracked)

Progress: 28%|██████| ETA: 0:13:00

8.800999 seconds (18.45 M allocations: 831.751 MiB, 4.10% gc time)
26 1.087219f0 (tracked)

Progress: 29%|██████████| ETA: 0:12:48

9.057523 seconds (18.45 M allocations: 831.642 MiB, 6.37% gc time)
27 1.0744874f0 (tracked)

Progress: 30%|██████████| ETA: 0:12:36

8.816793 seconds (18.45 M allocations: 831.770 MiB, 4.08% gc time)
28 1.070608f0 (tracked)

Progress: 31%|██████████| ETA: 0:12:24

8.757000 seconds (18.45 M allocations: 831.642 MiB, 4.16% gc time)
29 1.06775f0 (tracked)

Progress: 32%|██████████| ETA: 0:12:11

8.793988 seconds (18.45 M allocations: 831.751 MiB, 4.14% gc time)
30 1.0596961f0 (tracked)

Progress: 33%|██████████| ETA: 0:11:58

8.804556 seconds (18.45 M allocations: 831.650 MiB, 4.11% gc time)
31 1.0661788f0 (tracked)

Progress: 34%|██████████| ETA: 0:11:46

8.757650 seconds (18.45 M allocations: 831.751 MiB, 4.14% gc time)
32 1.0578994f0 (tracked)

Progress: 36%|██████████| ETA: 0:11:34

9.045728 seconds (18.45 M allocations: 831.643 MiB, 6.39% gc time)
33 1.0565302f0 (tracked)

Progress: 37%|██████████| ETA: 0:11:22

8.834256 seconds (18.45 M allocations: 831.762 MiB, 4.10% gc time)
34 1.0543725f0 (tracked)

Progress: 38%|██████████| ETA: 0:11:09

8.791016 seconds (18.45 M allocations: 831.642 MiB, 4.18% gc time)
35 1.0534203f0 (tracked)

Progress: 39%|██████████| ETA: 0:10:57

8.779481 seconds (18.45 M allocations: 831.751 MiB, 4.17% gc time)
36 1.0593368f0 (tracked)

Progress: 40%|██████████| ETA: 0:10:45

8.747900 seconds (18.45 M allocations: 831.650 MiB, 4.18% gc time)
37 1.053274f0 (tracked)

Progress: 41%|██████████| ETA: 0:10:33

8.790505 seconds (18.45 M allocations: 831.751 MiB, 4.17% gc time)

| ETA: 0:07:56

```
8.707062 seconds (18.45 M allocations: 831.636 MiB, 4.14% gc time)
```

| ETA: 0:07:44

8.709541 seconds (18.45 M allocations: 831.762 MiB, 4.15% gc time)

| ETA: 0:07:32

```
8.778030 seconds (18.45 M allocations: 831.636 MiB, 4.14% gc time)
```

| ETA: 0:07:20

```
8.821459 seconds (18.45 M allocations: 831.747 MiB, 4.09% gc time)
```

| ETA: 0:07:08

```
8.993877 seconds (18.45 M allocations: 831.645 MiB, 6.32% gc time)
```

| ETA: 0:06:56

8.720866 seconds (18.45 M allocations: 831.747 MiB, 4.15% gc time)

| ETA: 0:06:44

```
8.869076 seconds (18.45 M allocations: 831.643 MiB, 4.16% gc time)
```

| ETA: 0:06:32

```
8.714840 seconds (18.45 M allocations: 831.764 MiB, 4.16% gc time)
```

| ETA: 0:06:20

```
8.729166 seconds (18.45 M allocations: 831.633 MiB, 4.15% gc time)
```

| ETA: 0:06:08

```
8.736047 seconds (18.45 M allocations: 831.751 MiB, 4.12% gc time)
```

| ETA: 0:05:56

```
9.016458 seconds (18.45 M allocations: 831.656 MiB, 6.46% gc time)
```

| ETA: 0:05:44

```
8.718218 seconds (18.45 M allocations: 831.747 MiB, 4.11% gc time)
```

```
Progress:   69%|███████████                | ETA: 0:05:32
```

```
8.713232 seconds (18.45 M allocations: 831.642 MiB, 4.16% gc time)
63 1.0044646f0 (tracked)
```

```
Progress:   70%|███████████ | ETA: 0:05:20
```

```
8.735522 seconds (18.45 M allocations: 831.762 MiB, 4.13% gc time)
64 1.0015508f0 (tracked)
```

```
Progress: 71%|███████████████████████████████████████| ETA: 0:05:08
```

```
8.727698 seconds (18.45 M allocations: 831.642 MiB, 4.11% gc time)
65 0.9974902f0 (tracked)
```

```
Progress: 72%|███████████████████████████████████████| ETA: 0:04:56
```

```
8.744611 seconds (18.45 M allocations: 831.747 MiB, 4.11% gc time)
66 1.00382f0 (tracked)
```

```
Progress: 73%|███████████████████████████████████████| ETA: 0:04:45
```

```
8.727093 seconds (18.45 M allocations: 831.648 MiB, 4.15% gc time)
67 1.0029659f0 (tracked)
```

```
Progress:   74%|███████████                               | ETA: 0:04:33
```

```
8.809135 seconds (18.45 M allocations: 831.751 MiB, 4.17% gc time)
68 1.0015196f0 (tracked)
```

Progress: 76%|███████████████████████████████| ETA: 0:04:21

```
Progress: 76%|██████████████████████████████████████| ETA: 0:04:21
```

```
8.736342 seconds (18.45 M allocations: 831.642 MiB, 4.17% gc time)
69 1.0008705f0 (tracked)
```

Progress: 77%|███████████████████████████████| ETA: 0:04:09

```
Progress: 77%|██████████████████████████████████████| ETA: 0:04:09
```

8.750499 seconds (18.45 M allocations: 831.761 MiB, 4.16% gc time)

```
8.750499 seconds (18.45 M allocations: 831.761 MiB, 4.16% gc time)
70 0.999406f0 (tracked)
```

Progress: 78%|██████████████████████████████████████| ETA: 0:03:57

```
Progress: 78%|███████████████████████████████████████ | ETA: 0:03:57
```

```
8.707933 seconds (18.45 M allocations: 831.636 MiB, 4.16% gc time)
```

```
8.707933 seconds (18.45 M allocations: 831.636 MiB, 4.16% gc time)
71 0.99593693f0 (tracked)
```

Progress: 79%|███████████ | ETA: 0:03:45

```
Progress: 79%|███████████████████████████████████| ETA: 0:03:45
```

```
8.829688 seconds (18.45 M allocations; 831.747 MiB; 6.50% gc time)
```

```
8.929688 seconds (18.45 M allocations: 831.747 MiB, 6.50% gc time)
72 0.99304f0 (tracked)
```

Progress: 80%|██████████████████████████████████████| ETA: 0:03:33

```
Progress: 80%|███████████████████████████████████████████████████████████████████████████████| ETA: 0:03:33
```

8.734900 seconds (18.45 M allocations: 831.657 MiB, 4.17% gc time)

73 0.99550813f0 (tracked)

```
8.734900 seconds (18.45 M allocations: 831.657 MiB, 4.17% gc time)
73 0.99550813f0 (tracked)
```

Progress: 81%|██████████████████████████████████████| ETA: 0:03:21

```
Progress: 81%|██████████████████████████████████████████████████████████████████████████████| ETA: 0:03:21
```

```
8.785467 seconds (18.45 M allocations: 831.747 MiB, 4.20% gc time)
```

```
74 0.99797225f0 (tracked)
```

```
8.785467 seconds (18.45 M allocations: 831.747 MiB, 4.20% gc time)
74 0.99797225f0 (tracked)
```

Progress: 82%|███████████████████████████████████████| ETA: 0:03:09

Progress: 82%|███████████ | ETA: 0:03:09

[illegible][illegible]

```
Progress:   99%|███████████████████████████████████ | ETA: 0:00:12
```

[illegible]

```
function sample(input, len)
  m = cpu(model)
  # Resets all internal states.
  Flux.reset!(m)
  buf = IOBuffer()
  # Prime the model with our initial input.
  for c in input[1:end - 1]
```

```

        m(onehot(c, alphabet))
        write(buf, c)
    end
    write(buf, input[end])
    c = wsample(alphabet, m(onehot(input[end], alphabet)).data)
    write(buf, c)
    for i in 2:len
        c = wsample(alphabet, m(onehot(c, alphabet)).data)
        write(buf, c)
    end
    String(take!(buf))
end

```

Out[0]:

sample (generic function with 1 method)

In [0]:

```

println(sample("""Pirates of the Carribean""", 1204))
println(sample("""Joker is""", 1204))
println(sample("""This movie is""", 1204))
println(sample("""The latest James Bond movie""", 1204))
println(sample("""Woody Allen's film""", 1204))

```

Pirates of the Carribeand movies done , or makbert advisrated from Hollywood-like it , really inte
nascinating his curious and punt of lighty of prospires .

Joker is deeply watchable that from Gid .

This movie is simply out , longly humor .

The latest James Bond movie could 're left ' be the joible ... eleck classy and the nervally cinem
atic slaby-carffel .


```
Woody Allen's filmmakers hokey bodious worst than himself . .._ ' Welles Nichon is also are last t  
o too , Done Go solid .
```

In [0]:

In [0]:

```
%%shell
if ! command -v julia 2>&1 > /dev/null
then
    wget 'https://julialang-s3.julialang.org/bin/linux/x64/1.0/julia-1.0.5-linux-x86_64.tar.gz' \
        -O /tmp/julia.tar.gz
    tar -x -f /tmp/julia.tar.gz -C /usr/local --strip-components 1
    rm /tmp/julia.tar.gz
fi
julia -e "using Pkg; pkg\"add IJulia; precompile;\""

--2019-12-13 09:49:45-- https://julialang-s3.julialang.org/bin/linux/x64/1.0/julia-1.0.5-linux-x86_64.tar.gz
Resolving julialang-s3.julialang.org (julialang-s3.julialang.org)... 151.101.2.49, 151.101.66.49, 151.101.130.49, ...
Connecting to julialang-s3.julialang.org (julialang-s3.julialang.org)|151.101.2.49|:443...
connected.
HTTP request sent, awaiting response... 302 gce internal redirect trigger
Location: https://storage.googleapis.com/julialang2/bin/linux/x64/1.0/julia-1.0.5-linux-x86_64.tar.gz [following]
--2019-12-13 09:49:45-- https://storage.googleapis.com/julialang2/bin/linux/x64/1.0/julia-1.0.5-linux-x86_64.tar.gz
Resolving storage.googleapis.com (storage.googleapis.com)... 74.125.204.128, 2404:6800:4008:c00::80
Connecting to storage.googleapis.com (storage.googleapis.com)|74.125.204.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 88706549 (85M) [application/octet-stream]
Saving to: '/tmp/julia.tar.gz'

/tmp/julia.tar.gz 100%[=====>] 84.60M 21.8MB/s in 3.9s

2019-12-13 09:49:51 (21.8 MB/s) - '/tmp/julia.tar.gz' saved [88706549/88706549]

Cloning default registries into /root/.julia/registries
Cloning registry General from "https://github.com/JuliaRegistries/General.git"
Resolving package versions...
Installed VersionParsing — v1.2.0
Installed Conda — v1.3.0
Installed Parsers — v0.3.10
Installed ZMQ — v1.0.0
Installed BinaryProvider — v0.5.8
Installed JSON — v0.21.0
Installed IJulia — v1.20.2
Installed MbedTLS — v0.6.8
Installed SoftGlobalScope — v1.0.10
Updating `~/julia/environments/v1.0/Project.toml`
[7073ff75] + IJulia v1.20.2
Updating `~/julia/environments/v1.0/Manifest.toml`
[b99e7846] + BinaryProvider v0.5.8
[8f4d0f93] + Conda v1.3.0
[7073ff75] + IJulia v1.20.2
[682c06a0] + JSON v0.21.0
[739be429] + MbedTLS v0.6.8
[69de0a69] + Parsers v0.3.10
[b85f4697] + SoftGlobalScope v1.0.10
[81def892] + VersionParsing v1.2.0
[c2297ded] + ZMQ v1.0.0
[2a0f44e3] + Base64
[ade2ca70] + Dates
[8ba89e20] + Distributed
[7b1f6079] + FileWatching
[b77e0a4c] + InteractiveUtils
[76f85450] + LibGit2
[8f399da3] + Libdl
[37e2e46d] + LinearAlgebra
[56ddb016] + Logging
[d6f4376e] + Markdown
[a63ad114] + Mmap
[44cfe95a] + Pkg
[de0858da] + Printf
[3fa0cd96] + REPL
[9a3f8284] + Random
[8c564d8e] + SHA
```

```
[e80e919c] + SHA
[9e88b42a] + Serialization
[6462fe0b] + Sockets
[8dfed614] + Test
[cf7118a7] + UUIDs
[4ec0a83e] + Unicode
Building Conda → `~/.julia/packages/Conda/kLXeC/deps/build.log`
Building ZMQ → `~/.julia/packages/ZMQ/ABGOx/deps/build.log`
Building MbedTLS → `~/.julia/packages/MbedTLS/X4xar/deps/build.log`
Building IJulia → `~/.julia/packages/IJulia/F1GUo/deps/build.log`
Precompiling project...
Precompiling IJulia
```

Out[0]:

In [0]:

```
using Pkg

Pkg.add(Pkg.PackageSpec(;name="CuArrays", version=v"1.3.0"))
Pkg.add(Pkg.PackageSpec(;name="Flux", version=v"0.9.0"))

pkg"add Plots"
pkg"add StatsBase"
pkg"add Embeddings"
Pkg.add("DataFrames")
Pkg.add("CSV")
Pkg.add("ProgressBars")

pkg"precompile"

using CuArrays
using Flux
using Flux: onehot, chunk, batchseq, throttle, crossentropy
using StatsBase: wsample
using Embeddings

using Base.Iterators: partition
using LinearAlgebra
using Random
using Statistics
using Plots
using DataFrames
using CSV
using ProgressBars

Updating registry at `~/.julia/registries/General`
Updating git-repo `https://github.com/JuliaRegistries/General.git`
Resolving package versions...
Installed Requires ───────── v0.5.2
Installed Adapt ───────── v1.0.0
Installed Reexport ───────── v0.2.0
Installed GPUArrays ───────── v1.0.4
Installed MacroTools ───────── v0.5.3
Installed DataStructures ─── v0.17.6
Installed CuArrays ───────── v1.3.0
Installed FFTW ───────── v1.0.1
Installed CUDAapi ───────── v1.2.0
Installed AbstractFFTs ─── v0.4.1
Installed CEnum ───────── v0.2.0
Installed CUDAdrv ───────── v3.1.0
Installed NNlib ───────── v0.6.0
Installed TimerOutputs ─── v0.5.3
Installed CUDAnative ─── v2.4.0
Installed FillArrays ─── v0.7.4
Installed OrderedCollections ─ v1.1.0
Installed LLVM ───────── v1.3.2
Updating `~/.julia/environments/v1.0/Project.toml`
[3a865a2d] + CuArrays v1.3.0
Updating `~/.julia/environments/v1.0/Manifest.toml`
[621f4979] + AbstractFFTs v0.4.1
[79e6a3ab] + Adapt v1.0.0
[fa961155] + CEnum v0.2.0
[3895d2a7] + CUDAapi v1.2.0
[55f51811] + CUDAdrv v3.1.0
```

```

[0191614] + CUDAiv v0.1.0
[be33ccc6] + CUDAnative v2.4.0
[3a865a2d] + CuArrays v1.3.0
[864edb3b] + DataStructures v0.17.6
[7a1cc6ca] + FFTW v1.0.1
[1a297f60] + FillArrays v0.7.4
[0c68f7d7] + GPUArrays v1.0.4
[929cbde3] + LLVM v1.3.2
[1914dd2f] + MacroTools v0.5.3
[872c559c] + NNlib v0.6.0
[bac558e1] + OrderedCollections v1.1.0
[189a3867] + Reexport v0.2.0
[ae029012] + Requires v0.5.2
[a759f4b9] + TimerOutputs v0.5.3
[2f01184e] + SparseArrays
[10745b16] + Statistics
Building FFTW → ~/.julia/packages/FFTW/MJ7kl/deps/build.log`
Resolving package versions...
Installed NaNMath ————— v0.3.3
Installed DiffRules ————— v0.1.0
Installed AbstractTrees ————— v0.2.1
Installed FixedPointNumbers — v0.6.1
Installed BinDeps ————— v1.0.0
Installed CommonSubexpressions — v0.2.0
Installed Flux ————— v0.9.0
Installed ColorTypes ————— v0.8.0
Installed DataAPI ————— v1.1.0
Installed ZipFile ————— v0.8.3
Installed SpecialFunctions — v0.8.0
Installed Juno ————— v0.7.2
Installed DiffResults ————— v0.0.4
Installed URIParser ————— v0.4.0
Installed StaticArrays ————— v0.12.1
Installed Colors ————— v0.9.6
Installed Media ————— v0.5.0
Installed SortingAlgorithms — v0.3.1
Installed Missings ————— v0.4.3
Installed CodecZlib ————— v0.6.0
Installed Tracker ————— v0.2.5
Installed StatsBase ————— v0.32.0
Installed ForwardDiff ————— v0.10.7
Installed Compat ————— v2.2.0
Installed TranscodingStreams — v0.9.5
Updating ~/.julia/environments/v1.0/Project.toml`
[587475ba] + Flux v0.9.0
Updating ~/.julia/environments/v1.0/Manifest.toml`
[1520ce14] + AbstractTrees v0.2.1
[9e28174c] + BinDeps v1.0.0
[944b1d66] + CodecZlib v0.6.0
[3da002f7] + ColorTypes v0.8.0
[5ae59095] + Colors v0.9.6
[bbf7d656] + CommonSubexpressions v0.2.0
[34da2185] + Compat v2.2.0
[9a962f9c] + DataAPI v1.1.0
[163ba53b] + DiffResults v0.0.4
[b552c78f] + DiffRules v0.1.0
[53c48c17] + FixedPointNumbers v0.6.1
[587475ba] + Flux v0.9.0
[f6369f11] + ForwardDiff v0.10.7
[e5e0dc1b] + Juno v0.7.2
[e89f7d12] + Media v0.5.0
[e1d29d7a] + Missings v0.4.3
[77ba4419] + NaNMath v0.3.3
[a2af1166] + SortingAlgorithms v0.3.1
[276daf66] + SpecialFunctions v0.8.0
[90137ffa] + StaticArrays v0.12.1
[2913bbd2] + StatsBase v0.32.0
[9f7883ad] + Tracker v0.2.5
[3bb67fe8] + TranscodingStreams v0.9.5
[30578b45] + URIParser v0.4.0
[a5390f91] + ZipFile v0.8.3
[8bb1440f] + DelimitedFiles
[9abbd945] + Profile
[1a1011a3] + SharedArrays
Building SpecialFunctions → ~/.julia/packages/SpecialFunctions/ne2iw/deps/build.log`
Building ZipFile —————→ ~/.julia/packages/ZipFile/oD4uG/deps/build.log`
Building CodecZlib —————→ ~/.julia/packages/CodecZlib/5t9z0/deps/build.log`
Resolving package versions...

```

```

Resolving package versions...
Installed RecipesBase — v0.7.0
Installed Showoff — v0.3.1
Installed FFMPEG — v0.2.4
Installed Plots — v0.28.3
Installed Contour — v0.5.1
Installed PlotUtils — v0.6.1
Installed GeometryTypes — v0.7.6
Installed PlotThemes — v1.0.0
Installed Measures — v0.3.1
Installed IterTools — v1.3.0
Installed GR — v0.44.0
Updating `~/julia/environments/v1.0/Project.toml`
[91a5bcdd] + Plots v0.28.3
Updating `~/julia/environments/v1.0/Manifest.toml`
[d38c429a] + Contour v0.5.1
[c87230d0] + FFMPEG v0.2.4
[28b8d3ca] + GR v0.44.0
[4d00f742] + GeometryTypes v0.7.6
[c8e1da08] + IterTools v1.3.0
[442fdcdd] + Measures v0.3.1
[ccf2f8ad] + PlotThemes v1.0.0
[995b91a9] + PlotUtils v0.6.1
[91a5bcdd] + Plots v0.28.3
[3cdc5f2] + RecipesBase v0.7.0
[992d4aef] + Showoff v0.3.1
Building GR —→ `~/julia/packages/GR/oiZD3/deps/build.log`
Building FFMPEG → `~/julia/packages/FFMPEG/guNlx/deps/build.log`
Building Plots → `~/julia/packages/Plots/RsO3g/deps/build.log`
Resolving package versions...
Updating `~/julia/environments/v1.0/Project.toml`
[2913bbd2] + StatsBase v0.32.0
Updating `~/julia/environments/v1.0/Manifest.toml`
[no changes]
Resolving package versions...
Installed IniFile — v0.5.0
Installed AutoHashEquals — v0.2.0
Installed Embeddings — v0.4.1
Installed DataDeps — v0.7.1
Installed HTTP — v0.8.8
Updating `~/julia/environments/v1.0/Project.toml`
[c5bfea45] + Embeddings v0.4.1
Updating `~/julia/environments/v1.0/Manifest.toml`
[15f4f7f2] + AutoHashEquals v0.2.0
[124859b0] + DataDeps v0.7.1
[c5bfea45] + Embeddings v0.4.1
[cd3eb016] + HTTP v0.8.8
[83e8ac13] + IniFile v0.5.0
Resolving package versions...
Installed IteratorInterfaceExtensions — v1.0.0
Installed InvertedIndices — v1.0.0
Installed PooledArrays — v0.5.2
Installed DataValueInterfaces — v1.0.0
Installed TableTraits — v1.0.0
Installed Tables — v0.2.11
Installed CategoricalArrays — v0.7.4
Installed DataFrames — v0.20.0
Updating `~/julia/environments/v1.0/Project.toml`
[a93c6f00] + DataFrames v0.20.0
Updating `~/julia/environments/v1.0/Manifest.toml`
[324d7699] + CategoricalArrays v0.7.4
[a93c6f00] + DataFrames v0.20.0
[e2d170a0] + DataValueInterfaces v1.0.0
[41ab1584] + InvertedIndices v1.0.0
[82899510] + IteratorInterfaceExtensions v1.0.0
[2dfb63ee] + PooledArrays v0.5.2
[3783bdb8] + TableTraits v1.0.0
[bd369af6] + Tables v0.2.11
[9fa8497b] + Future
Resolving package versions...
Installed WeakRefStrings — v0.6.1
Installed FilePathsBase — v0.7.0
Installed LazyArrays — v0.13.1
Installed CSV — v0.5.18
Updating `~/julia/environments/v1.0/Project.toml`
[336ed68f] + CSV v0.5.18
Updating `~/julia/environments/v1.0/Manifest.toml`
[336ed68f] + CSV v0.5.18

```

```
[336ed68f] + CSV v0.5.18
[48062228] + FilePathsBase v0.7.0
[5078a376] + LazyArrays v0.13.1
[eal0d353] + WeakRefStrings v0.6.1
Resolving package versions...
Installed ProgressBars - v0.3.2
Updating `~/julia/environments/v1.0/Project.toml`
[49802e3a] + ProgressBars v0.3.2
Updating `~/julia/environments/v1.0/Manifest.toml`
[49802e3a] + ProgressBars v0.3.2
Precompiling project...
Precompiling CSV
```

```
[ Info: Precompiling CSV [336ed68f-0bac-5ca0-87d4-7b16caf5d00b]
└ @ Base loading.jl:1192
```

Precompiling StatsBase

```
[ Info: Precompiling StatsBase [2913bbd2-ae8a-5f71-8c99-4fb6c76f3a91]
└ @ Base loading.jl:1192
```

Precompiling CuArrays

```
[ Info: Precompiling CuArrays [3a865a2d-5b23-5a0f-bc46-62713ec82fae]
└ @ Base loading.jl:1192
```

Precompiling Flux

```
[ Info: Precompiling Flux [587475ba-b771-5e3f-ad9e-33799f191a9c]
└ @ Base loading.jl:1192
Internal error: encountered unexpected error in runtime:
BoundsError(a=Array{Core.Compiler.BasicBlock, (32,)}[
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=1, last=6), preds=Array{Int64, (1,)}[32], succs=Array{Int64, (1,)}[2]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=7, last=13), preds=Array{Int64, (1,)}[1], succs=Array{Int64, (2,)}[5, 3]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=14, last=19), preds=Array{Int64, (1,)}[2], succs=Array{Int64, (1,)}[4]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=20, last=20), preds=Array{Int64, (1,)}[3], succs=Array{Int64, (1,)}[7]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=21, last=27), preds=Array{Int64, (1,)}[2], succs=Array{Int64, (1,)}[6]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=28, last=28), preds=Array{Int64, (1,)}[5], succs=Array{Int64, (1,)}[7]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=29, last=46), preds=Array{Int64, (2,)}[4, 6], succs=Array{Int64, (2,)}[9, 8]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=47, last=48), preds=Array{Int64, (1,)}[7], succs=Array{Int64, (0,)}[]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=49, last=49), preds=Array{Int64, (1,)}[7], succs=Array{Int64, (1,)}[10]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=50, last=51), preds=Array{Int64, (1,)}[9], succs=Array{Int64, (1,)}[11]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=52, last=52), preds=Array{Int64, (1,)}[10], succs=Array{Int64, (1,)}[12]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=53, last=53), preds=Array{Int64, (1,)}[11], succs=Array{Int64, (1,)}[13]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=54, last=55), preds=Array{Int64, (1,)}[12], succs=Array{Int64, (1,)}[14]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=56, last=64), preds=Array{Int64, (1,)}[13], succs=Array{Int64, (1,)}[15]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=65, last=69), preds=Array{Int64, (1,)}[14], succs=Array{Int64, (2,)}[17, 16]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=70, last=72), preds=Array{Int64, (1,)}[15], succs=Array{Int64, (0,)}[]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=73, last=80), preds=Array{Int64, (1,)}[15], succs=Array{Int64, (2,)}[19, 18]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=81, last=82), preds=Array{Int64, (1,)}[17], succs=Array{Int64, (0,)}[]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=83, last=83), preds=Array{Int64, (1,)}[17], succs=Array{Int64, (1,)}[20]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=84, last=85), preds=Array{Int64, (1,)}[19], succs=Array{Int64, (1,)}[21]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=86, last=86), preds=Array{Int64,
```

```

(1,))[20], succs=Array{Int64, (1,)}[22]],
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=87, last=87), preds=Array{Int64,
(1,)}[21], succs=Array{Int64, (1,)}[23]],
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=88, last=88), preds=Array{Int64,
(1,)}[22], succs=Array{Int64, (1,)}[24]],
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=89, last=90), preds=Array{Int64,
(1,)}[23], succs=Array{Int64, (1,)}[25]],
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=91, last=91), preds=Array{Int64,
(1,)}[24], succs=Array{Int64, (1,)}[26]],
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=92, last=92), preds=Array{Int64,
(1,)}[25], succs=Array{Int64, (1,)}[27]],
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=93, last=93), preds=Array{Int64,
(1,)}[26], succs=Array{Int64, (2,)}[29, 28]],
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=94, last=96), preds=Array{Int64,
(1,)}[27], succs=Array{Int64, (0,)}[]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=97, last=100), preds=Array{Int64,
(1,)}[27], succs=Array{Int64, (2,)}[31, 30]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=101, last=101), preds=Array{Int64,
(1,)}[29], succs=Array{Int64, (1,)}[32]],
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=102, last=102), preds=Array{Int64,
(1,)}[29], succs=Array{Int64, (1,)}[32]],
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=103, last=105), preds=Array{Int64,
(2,)}[30, 31], succs=Array{Int64, (1,)}[1]]], i=(0,))
rec_backtrace at /buildworker/worker/package_linux64/build/src/stackwalk.c:94
record_backtrace at /buildworker/worker/package_linux64/build/src/task.c:246
jl_throw at /buildworker/worker/package_linux64/build/src/task.c:577
jl_bounds_error_ints at /buildworker/worker/package_linux64/build/src/rtutils.c:187
getindex at ./array.jl:731
jffptr_getindex_2271.clone_1 at /usr/local/lib/julia/sys.so (unknown line)
jl_apply_generic at /buildworker/worker/package_linux64/build/src/gf.c:2162
replace_code_newstyle! at ./compiler/ssair/legacy.jl:86
optimize at ./compiler/optimize.jl:212
typeinf at ./compiler/typeinfer.jl:35
typeinf_ext at ./compiler/typeinfer.jl:574
typeinf_ext at ./compiler/typeinfer.jl:611
jffptr_typeinf_ext_1.clone_1 at /usr/local/lib/julia/sys.so (unknown line)
jl_apply_generic at /buildworker/worker/package_linux64/build/src/gf.c:2162
jl_apply at /buildworker/worker/package_linux64/build/src/julia.h:1537 [inlined]
jl_apply_with_saved_exception_state at /buildworker/worker/package_linux64/build/src/rtutils.c:257
jl_type_infer at /buildworker/worker/package_linux64/build/src/gf.c:253
jl_compile_method_internal at /buildworker/worker/package_linux64/build/src/gf.c:1764 [inlined]
jl_fptr_trampoline at /buildworker/worker/package_linux64/build/src/gf.c:1808
jl_apply_generic at /buildworker/worker/package_linux64/build/src/gf.c:2162
jl_apply at /buildworker/worker/package_linux64/build/src/julia.h:1537 [inlined]
start_task at /buildworker/worker/package_linux64/build/src/task.c:268
unknown function (ip: 0xffffffffffffffff)

```

Precompiling Embeddings

```

└ Info: Precompiling Embeddings [c5bfea45-b7f1-5224-a596-15500f5db411]
└ @ Base loading.jl:1192

```

Precompiling Plots

```

└ Info: Precompiling Plots [91a5bcdd-55d7-5caf-9e0b-520d859cae80]
└ @ Base loading.jl:1192

```

Precompiling ProgressBars

```

└ Info: Precompiling ProgressBars [49802e3a-d2f1-5c88-81d8-b72133a6f568]
└ @ Base loading.jl:1192
Internal error: encountered unexpected error in runtime:
BoundsError{a=Array{Core.Compiler.BasicBlock, (32,)}[
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=1, last=6), preds=Array{Int64,
(1,)}[32], succs=Array{Int64, (1,)}[2]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=7, last=13), preds=Array{Int64,
(1,)}[1], succs=Array{Int64, (2,)}[5, 3]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=14, last=19), preds=Array{Int64,
(1,)}[2], succs=Array{Int64, (1,)}[4]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=20, last=20), preds=Array{Int64,
(1,)}[3], succs=Array{Int64, (1,)}[7]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=21, last=27), preds=Array{Int64,
(1,)}[2], succs=Array{Int64, (1,)}[6])],

```



```

Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=28, last=28), preds=Array{Int64,
(1,)}[5], succs=Array{Int64, (1,)}[7]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=29, last=46), preds=Array{Int64,
(2,)}[4, 6], succs=Array{Int64, (2,)}[9, 8]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=47, last=48), preds=Array{Int64,
(1,)}[7], succs=Array{Int64, (0,)}[]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=49, last=49), preds=Array{Int64,
(1,)}[7], succs=Array{Int64, (1,)}[10]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=50, last=51), preds=Array{Int64,
(1,)}[9], succs=Array{Int64, (1,)}[11]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=52, last=52), preds=Array{Int64,
(1,)}[10], succs=Array{Int64, (1,)}[12]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=53, last=53), preds=Array{Int64,
(1,)}[11], succs=Array{Int64, (1,)}[13]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=54, last=55), preds=Array{Int64,
(1,)}[12], succs=Array{Int64, (1,)}[14]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=56, last=64), preds=Array{Int64,
(1,)}[13], succs=Array{Int64, (1,)}[15]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=65, last=69), preds=Array{Int64,
(1,)}[14], succs=Array{Int64, (2,)}[17, 16]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=70, last=72), preds=Array{Int64,
(1,)}[15], succs=Array{Int64, (0,)}[]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=73, last=80), preds=Array{Int64,
(1,)}[15], succs=Array{Int64, (2,)}[19, 18]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=81, last=82), preds=Array{Int64,
(1,)}[17], succs=Array{Int64, (0,)}[]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=83, last=83), preds=Array{Int64,
(1,)}[17], succs=Array{Int64, (1,)}[20]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=84, last=85), preds=Array{Int64,
(1,)}[19], succs=Array{Int64, (1,)}[21]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=86, last=86), preds=Array{Int64,
(1,)}[20], succs=Array{Int64, (1,)}[22]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=87, last=87), preds=Array{Int64,
(1,)}[21], succs=Array{Int64, (1,)}[23]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=88, last=88), preds=Array{Int64,
(1,)}[22], succs=Array{Int64, (1,)}[24]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=89, last=90), preds=Array{Int64,
(1,)}[23], succs=Array{Int64, (1,)}[25]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=91, last=91), preds=Array{Int64,
(1,)}[24], succs=Array{Int64, (1,)}[26]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=92, last=92), preds=Array{Int64,
(1,)}[25], succs=Array{Int64, (1,)}[27]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=93, last=93), preds=Array{Int64,
(1,)}[26], succs=Array{Int64, (2,)}[29, 28]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=94, last=96), preds=Array{Int64,
(1,)}[27], succs=Array{Int64, (0,)}[]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=97, last=100), preds=Array{Int64,
(1,)}[27], succs=Array{Int64, (2,)}[31, 30]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=101, last=101), preds=Array{Int64,
(1,)}[29], succs=Array{Int64, (1,)}[32]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=102, last=102), preds=Array{Int64,
(1,)}[29], succs=Array{Int64, (1,)}[32]),
Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=103, last=105), preds=Array{Int64,
(2,)}[30, 31], succs=Array{Int64, (1,)}[1]}, i=(0,))
rec_backtrace at /buildworker/worker/package_linux64/build/src/stackwalk.c:94
record_backtrace at /buildworker/worker/package_linux64/build/src/task.c:246
jl_throw at /buildworker/worker/package_linux64/build/src/task.c:577
jl_bounds_error_ints at /buildworker/worker/package_linux64/build/src/rtutils.c:187
getindex at ./array.jl:731
jfp_ptr_getindex_2271.clone_1 at /usr/local/lib/julia/sys.so (unknown line)
jl_apply_generic at /buildworker/worker/package_linux64/build/src/gf.c:2162
replace_code_newstyle! at ./compiler/ssair/legacy.jl:86
optimize at ./compiler/optimize.jl:212
typeinf at ./compiler/typeinfer.jl:35
typeinf_ext at ./compiler/typeinfer.jl:574
typeinf_ext at ./compiler/typeinfer.jl:611
jfp_ptr_typeinf_ext_1.clone_1 at /usr/local/lib/julia/sys.so (unknown line)
jl_apply_generic at /buildworker/worker/package_linux64/build/src/gf.c:2162
jl_apply at /buildworker/worker/package_linux64/build/src/julia.h:1537 [inlined]
jl_apply_with_saved_exception_state at /buildworker/worker/package_linux64/build/src/rtutils.c:257
jl_type_infer at /buildworker/worker/package_linux64/build/src/gf.c:253
jl_compile_method_internal at /buildworker/worker/package_linux64/build/src/gf.c:1764 [inlined]
jl_fptr_trampoline at /buildworker/worker/package_linux64/build/src/gf.c:1808
jl_apply_generic at /buildworker/worker/package_linux64/build/src/gf.c:2162
jl_apply at /buildworker/worker/package_linux64/build/src/julia.h:1537 [inlined]
start_task at /buildworker/worker/package_linux64/build/src/task.c:268
unknown function (ip: 0xffffffffffffffff)

```


Assignment 2: “Half-bag”

2.4: Bag of vectors

Data

Stanford Sentiment Treebank

In [0]:

```
run(`curl -fsS https://nlp.stanford.edu/sentiment/trainDevTestTrees_PTB.zip -o
/tmp/trainDevTestTrees_PTB.zip`)
run(`unzip -q -o -d /tmp /tmp/trainDevTestTrees_PTB.zip`)
run(`rm -f /tmp/trainDevTestTrees_PTB.zip`)

nothing
```

In [0]:

```
function loadsst(path)
  xs = Array{String}[]
  ys = Int[]
  open(path) do file
    # Quick, dirty, and improper S-expression parsing.
    for line in eachline(file)
      soup = split(line)
      push!(ys, parse{Int, lstrip(first(soup), '(')})
      tokens = String[]
      for chunk in soup[2:end]
        endswith(chunk, ")") || continue
        push!(tokens, rstrip(chunk, ')'))
      end
      push!(xs, tokens)
    end
  end
  xs, ys
end

ssttrainxs, ssttrainys = loadsst("/tmp/trees/train.txt")
sstvalidxs, sstvalidys = loadsst("/tmp/trees/dev.txt")
ssttestxs, ssttestys = loadsst("/tmp/trees/test.txt")

nothing
```

In [0]:

```
for _ in 1:32
  i = rand(1:length(ssttrainxs))
  println("$ssttrainys[i]: $(join(ssttrainxs[i], ' '))")
end
```

4: Spiderman ROCKS
2: ... gripping and handsome execution , -LRB- but -RRB- there is n't much about K-19 that 's unique or memorable .
1: The film has the thrown-together feel of a summer-camp talent show : hastily written , underrehearsed , arbitrarily plotted and filled with crude humor and vulgar innuendo .
3: Oh , James !
2: With three excellent principal singers , a youthful and good-looking diva and tenor and richly handsome locations , it 's enough to make you wish Jacquot had left well enough alone and just filmed the opera without all these distortions of perspective .
2: This film was made by and for those folks who collect the serial killer cards and are fascinated by the mere suggestion of serial killers .
0: The exclamation point seems to be the only bit of glee you 'll find in this dreary mess .
0: A movie more to be prescribed than recommended -- as visually bland as a dentist 's waiting room , complete with soothing Muzak and a cushion of predictable narrative rhythms .
1: Every joke is repeated at least four times .

2: `` Sum `` is Jack Ryan 's `` do-over . ``

3: There 's absolutely no reason why Blue Crush , a late-summer surfer girl entry , should be as e
ntertaining as it is

0: We never truly come to care about the main characters and whether or not they 'll wind up
together , and Michele 's spiritual quest is neither amusing nor dramatic enough to sustain
interest .

3: Although Estela Bravo 's documentary is cloyingly hagiographic in its portrait of Cuban leader
Fidel Castro , it 's still a guilty pleasure to watch .

3: A metaphor for a modern-day urban China searching for its identity .

3: The locations go from stark desert to gorgeous beaches .

1: The story alone could force you to scratch a hole in your head .

0: ... a sour little movie at its core ; an exploration of the emptiness that underlay the relentl
ess gaiety of the 1920 's ... The film 's ending has a `` What was it all for ? ``

2: Its juxtaposition of overwrought existentialism and stomach-churning gore will have you forever
on the verge of either cracking up or throwing up .

4: A delicious and delicately funny look at the residents of a Copenhagen neighborhood coping with
the befuddling complications life tosses at them .

1: Has all the hallmarks of a movie designed strictly for children 's home video , a market so ins
atiable it absorbs all manner of lame entertainment , as long as 3-year-olds find it diverting .

3: ... manages to deliver a fair bit of vampire fun .

0: Priggish , lethargically paced parable of renewal .

4: This documentary is a dazzling , remarkably unpretentious reminder of what -LRB- Evans -RRB- ha
d , lost , and got back .

4: likeable thanks to its cast , its cuisine and its quirky tunes .

1: Each scene drags , underscoring the obvious , and sentiment is slathered on top .

2: A behind the scenes look at the training and dedication that goes into becoming a world-class f
encer and the champion that 's made a difference to NYC inner-city youth .

3: Writer \/ director M. Night Shyamalan 's ability to pull together easily accessible stories tha
t resonate with profundity is undeniable .

1: Sorvino makes the princess seem smug and cartoonish , and the film only really comes alive when
poor Hermocrates and Leontine pathetically compare notes about their budding amours .

1: Well , this movie proves you wrong on both counts .

1: It 's like going to a house party and watching the host defend himself against a frothing ex-gi
rlfriend .

1: Not a strike against Yang 's similarly themed Yi Yi , but I found What Time ?

3: Based on Dave Barry 's popular book of the same name , the movie benefits from having a real wr
iter plot out all of the characters ' moves and overlapping story .

TrainingState struct used in callbacks to record training state:

In [0]:

```
mutable struct TrainingState_
    train_losses::Vector{Float64}
    valid_losses::Vector{Float64}
end
TrainingState_() = TrainingState_([], [])

function update_loss!(state::TrainingState_, loss)
    push!(state.train_losses, Tracker.data(loss(trainxs, trainys)))
    push!(state.valid_losses, Tracker.data(loss(validxs, validys)))
end;
```

Utility functions:

- Accuracy function

In [0]:

```
accuracy(y, y_preds) = 100*(mean(y .== y_preds));
```

- Prediction function

In [0]:

```
function max_p(distr)
    preds = Vector{Int64}()
    for i in 1:size(distr)[2]
        push!(preds, argmax(distr[:, i]))
    end
```

```
    return preds .- 1;
end;
```

- Activation function (logistic)

In [0]:

```
logistic(x) = 1/(1 .+ exp.(-x));
```

4.1 Implement a bag of vectors model with a cross-entropy loss for the Stanford Sentiment Treebank data, based on the pre-trained word2vec embeddings provided in the Assignment Colaboratory notebook.

Embeddings:

In [0]:

```
const w2v          = load_embeddings(Word2Vec)
const token2index = Dict{Token => Int} for (i, token) in enumerate(w2v.vocab)
function embedding(token)
    # If a token is not in our vocabulary, we use "UNK" to represent it.
    token = token in w2v.vocab ? token : "UNK"
    w2v.embeddings[:, token2index[token]]
end

nothing
```

This program has requested access to the data dependency word2vec 300d. which is not currently installed. It can be installed automatically, and you will not see this message again.

Pretrained Word2Vec Word embeddings
Website: <https://code.google.com/archive/p/word2vec/>
Author: Mikolov et al.
Year: 2013

Pre-trained vectors trained on part of Google News dataset (about 100 billion words). The model contains 300-dimensional vectors for 3 million words and phrases.

Paper:

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. In Proceedings of NIPS, 2013.

Do you want to download the dataset from <https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz> to "/root/.julia/datadeps/word2vec 300d"?

[y/n]

stdin> y

```
Info: Downloading
| source = https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz
| dest = /root/.julia/datadeps/word2vec 300d/GoogleNews-vectors-negative300.bin.gz
| progress = 0.0425
| time_taken = 5.03 s
| time_remaining = 113.48 s
| average_speed = 13.253 MiB/s
| downloaded = 66.704 MiB
| remaining = 1.469 GiB
| total = 1.534 GiB
└ @ HTTP /root/.julia/packages/HTTP/lZVI1/src/download.jl:119
Info: Downloading
| source = https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz
| dest = /root/.julia/datadeps/word2vec 300d/GoogleNews-vectors-negative300.bin.gz
| progress = 0.0902
| time_taken = 10.07 s
| time_remaining = 101.64 s
| average_speed = 14.060 MiB/s
| downloaded = 141.616 MiB
| remaining = 1.396 GiB
```

```
| total = 1.534 GiB
| @ HTTP /root/.julia/packages/HTTP/lZVI1/src/download.jl:119
| Info: Downloading
|   source = https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz
|   dest = /root/.julia/datadeps/word2vec 300d/GoogleNews-vectors-negative300.bin.gz
|   progress = 0.1425
|   time_taken = 15.07 s
|   time_remaining = 90.71 s
|   average_speed = 14.849 MiB/s
|   downloaded = 223.820 MiB
|   remaining = 1.315 GiB
|   total = 1.534 GiB
| @ HTTP /root/.julia/packages/HTTP/lZVI1/src/download.jl:119
| Info: Downloading
|   source = https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz
|   dest = /root/.julia/datadeps/word2vec 300d/GoogleNews-vectors-negative300.bin.gz
|   progress = 0.1949
|   time_taken = 20.12 s
|   time_remaining = 83.11 s
|   average_speed = 15.215 MiB/s
|   downloaded = 306.179 MiB
|   remaining = 1.235 GiB
|   total = 1.534 GiB
| @ HTTP /root/.julia/packages/HTTP/lZVI1/src/download.jl:119
| Info: Downloading
|   source = https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz
|   dest = /root/.julia/datadeps/word2vec 300d/GoogleNews-vectors-negative300.bin.gz
|   progress = 0.2472
|   time_taken = 25.12 s
|   time_remaining = 76.53 s
|   average_speed = 15.452 MiB/s
|   downloaded = 388.226 MiB
|   remaining = 1.155 GiB
|   total = 1.534 GiB
| @ HTTP /root/.julia/packages/HTTP/lZVI1/src/download.jl:119
| Info: Downloading
|   source = https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz
|   dest = /root/.julia/datadeps/word2vec 300d/GoogleNews-vectors-negative300.bin.gz
|   progress = 0.2996
|   time_taken = 30.12 s
|   time_remaining = 70.44 s
|   average_speed = 15.619 MiB/s
|   downloaded = 470.523 MiB
|   remaining = 1.074 GiB
|   total = 1.534 GiB
| @ HTTP /root/.julia/packages/HTTP/lZVI1/src/download.jl:119
| Info: Downloading
|   source = https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz
|   dest = /root/.julia/datadeps/word2vec 300d/GoogleNews-vectors-negative300.bin.gz
|   progress = 0.3512
|   time_taken = 35.13 s
|   time_remaining = 64.9 s
|   average_speed = 15.703 MiB/s
|   downloaded = 551.601 MiB
|   remaining = 1019.145 MiB
|   total = 1.534 GiB
| @ HTTP /root/.julia/packages/HTTP/lZVI1/src/download.jl:119
| Info: Downloading
|   source = https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz
|   dest = /root/.julia/datadeps/word2vec 300d/GoogleNews-vectors-negative300.bin.gz
|   progress = 0.4034
|   time_taken = 40.13 s
|   time_remaining = 59.35 s
|   average_speed = 15.790 MiB/s
|   downloaded = 633.616 MiB
|   remaining = 937.129 MiB
|   total = 1.534 GiB
| @ HTTP /root/.julia/packages/HTTP/lZVI1/src/download.jl:119
| Info: Downloading
|   source = https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz
|   dest = /root/.julia/datadeps/word2vec 300d/GoogleNews-vectors-negative300.bin.gz
|   progress = 0.4556
|   time_taken = 45.13 s
|   time_remaining = 53.92 s
|   average_speed = 15.858 MiB/s
|   downloaded = 715.648 MiB
|   remaining = 855.098 MiB
```

```
total = 1.534 GiB
└ @ HTTP /root/.julia/packages/HTTP/lZVI1/src/download.jl:119
└ Info: Downloading
  source = https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz
  dest = /root/.julia/datadeps/word2vec 300d/GoogleNews-vectors-negative300.bin.gz
  progress = 0.5073
  time_taken = 50.13 s
  time_remaining = 48.68 s
  average_speed = 15.896 MiB/s
  downloaded = 796.866 MiB
  remaining = 773.879 MiB
  total = 1.534 GiB
└ @ HTTP /root/.julia/packages/HTTP/lZVI1/src/download.jl:119
└ Info: Downloading
  source = https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz
  dest = /root/.julia/datadeps/word2vec 300d/GoogleNews-vectors-negative300.bin.gz
  progress = 0.5596
  time_taken = 55.13 s
  time_remaining = 43.39 s
  average_speed = 15.943 MiB/s
  downloaded = 878.976 MiB
  remaining = 691.770 MiB
  total = 1.534 GiB
└ @ HTTP /root/.julia/packages/HTTP/lZVI1/src/download.jl:119
└ Info: Downloading
  source = https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz
  dest = /root/.julia/datadeps/word2vec 300d/GoogleNews-vectors-negative300.bin.gz
  progress = 0.6117
  time_taken = 60.13 s
  time_remaining = 38.18 s
  average_speed = 15.978 MiB/s
  downloaded = 960.788 MiB
  remaining = 609.957 MiB
  total = 1.534 GiB
└ @ HTTP /root/.julia/packages/HTTP/lZVI1/src/download.jl:119
└ Info: Downloading
  source = https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz
  dest = /root/.julia/datadeps/word2vec 300d/GoogleNews-vectors-negative300.bin.gz
  progress = 0.6635
  time_taken = 65.14 s
  time_remaining = 33.04 s
  average_speed = 15.998 MiB/s
  downloaded = 1.018 GiB
  remaining = 528.535 MiB
  total = 1.534 GiB
└ @ HTTP /root/.julia/packages/HTTP/lZVI1/src/download.jl:119
└ Info: Downloading
  source = https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz
  dest = /root/.julia/datadeps/word2vec 300d/GoogleNews-vectors-negative300.bin.gz
  progress = 0.7156
  time_taken = 70.15 s
  time_remaining = 27.88 s
  average_speed = 16.024 MiB/s
  downloaded = 1.098 GiB
  remaining = 446.707 MiB
  total = 1.534 GiB
└ @ HTTP /root/.julia/packages/HTTP/lZVI1/src/download.jl:119
└ Info: Downloading
  source = https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz
  dest = /root/.julia/datadeps/word2vec 300d/GoogleNews-vectors-negative300.bin.gz
  progress = 0.7678
  time_taken = 75.15 s
  time_remaining = 22.72 s
  average_speed = 16.049 MiB/s
  downloaded = 1.178 GiB
  remaining = 364.692 MiB
  total = 1.534 GiB
└ @ HTTP /root/.julia/packages/HTTP/lZVI1/src/download.jl:119
└ Info: Downloading
  source = https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz
  dest = /root/.julia/datadeps/word2vec 300d/GoogleNews-vectors-negative300.bin.gz
  progress = 0.8197
  time_taken = 80.16 s
  time_remaining = 17.63 s
  average_speed = 16.063 MiB/s
  downloaded = 1.257 GiB
  remaining = 283.192 MiB
```

```

total = 1.534 GiB
@ HTTP /root/.julia/packages/HTTP/lZVI1/src/download.jl:119
Info: Downloading
source = https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz
dest = /root/.julia/datadeps/word2vec 300d/GoogleNews-vectors-negative300.bin.gz
progress = 0.8717
time_taken = 85.16 s
time_remaining = 12.53 s
average_speed = 16.079 MiB/s
downloaded = 1.337 GiB
remaining = 201.504 MiB
total = 1.534 GiB
@ HTTP /root/.julia/packages/HTTP/lZVI1/src/download.jl:119
Info: Downloading
source = https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz
dest = /root/.julia/datadeps/word2vec 300d/GoogleNews-vectors-negative300.bin.gz
progress = 0.9239
time_taken = 90.16 s
time_remaining = 7.42 s
average_speed = 16.096 MiB/s
downloaded = 1.417 GiB
remaining = 119.504 MiB
total = 1.534 GiB
@ HTTP /root/.julia/packages/HTTP/lZVI1/src/download.jl:119
Info: Downloading
source = https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz
dest = /root/.julia/datadeps/word2vec 300d/GoogleNews-vectors-negative300.bin.gz
progress = 0.9759
time_taken = 95.16 s
time_remaining = 2.35 s
average_speed = 16.108 MiB/s
downloaded = 1.497 GiB
remaining = 37.864 MiB
total = 1.534 GiB
@ HTTP /root/.julia/packages/HTTP/lZVI1/src/download.jl:119
Info: Downloading
source = https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz
dest = /root/.julia/datadeps/word2vec 300d/GoogleNews-vectors-negative300.bin.gz
progress = 1.0
time_taken = 97.49 s
time_remaining = 0.0 s
average_speed = 16.112 MiB/s
downloaded = 1.534 GiB
remaining = 0 bytes
total = 1.534 GiB
@ HTTP /root/.julia/packages/HTTP/lZVI1/src/download.jl:119
tcmmalloc: large alloc 3600007168 bytes == 0x10750000 @ 0x7fefd528cb6b 0x7fefd52ac379
0x7fefd4b9f5ec 0x7fefd4b6942b 0x7fefd4b57cda4a 0x7fefd4b57cca3c 0x7fefd4b4805c 0x7fefd4b4cca7
0x7fefd4b57cc598 0x7fefd4b57cc2f0 0x7fefd4b57cc3d8 0x7fefd4b4cca7 0x7fefd4b57832d9 0x7fefd4b578339b
0x7fefd4b4cca7 0x7fefd4b578307a 0x7fefd4b4c0d6 0x7fefd4cb5f10 0x7fefd4cb5c39 0x7fefd4cb65bc
0x7fefd4cb6d3f 0x7fefd4b6296c 0x7fefd4cb780d 0x7fefd4b81ffc 0x7fefd4b5d8e0 0x7fefd4b5732f64
0x7fefd4b4c0d6 0x7fefd4b572d7e7 0x7fefd4b4c0d6 0x7fefd4b5a846 0x7fefd4b5aea2

```

Function to process the sentences and apply pooling operations: Each word of each sentence is first mapped to an embedding which creates a $300 \times (\text{number of words in that sentence})$ matrix for each sentence. Then each matrix is mapped to a 300×1 vector by applying max pooling, mean pooling, or both.

In [0]:

```

function sentence_processing(xs)
    xs_ = []
    for j = 1:size(xs)[1]
        n = size(xs[j])[1]
        matrix = zeros(300, n)
        i = 1
        for token in xs[j]
            matrix[:,i] = embedding(token)
            i += 1
        end
        push!(xs_, matrix)
    end

    N = size(xs_)[1] # Mean &/or Max pool
    xs_mean = zeros((300,N))
    #xs_max = zeros((300,N))

```

```

for j = 1:N
    for i in 1:300
        xs_mean[i,j] = mean(xs_[j][i,:])
        #xs_max[i,j] = maximum(xs_[j][i,:])
    end
end

#output = vcat(xs_mean, xs_max);
output = xs_mean

return output
end

```

Out[0]:

sentence_processing (generic function with 1 method)

Create Training, validation and test data:

In [0]:

```

using Flux: onehotbatch

trainxs = sentence_processing(ssttrainxs) |> gpu ;
trainys = onehotbatch(ssttrainys, 0:4) |> gpu ;
validxs = sentence_processing(sstvalidxs) |> gpu ;
validys = onehotbatch(sstvalidys, 0:4) |> gpu ;
testxs = sentence_processing(ssttestxs) |> gpu ;
testys = onehotbatch(ssttestys, 0:4) |> gpu ;

```

Define model and training: Here we use the general constructor of an MLP as in Question 1, and use the processed data from above as the inputs.

In [0]:

```

# Generic function for building arbitrarily sized mlp
# Activation functions are fixed to be sigmoid except for last layer's softmax

function build_mlp(input_size::Int64, output_size::Int64, hids::Vector{Int64})
    #ensure this function is not being passed nonsense
    @assert length(hids) > 0
    @assert all(hids .> 0)

    layer_list = []
    push!(layer_list, Dense(input_size, hids[1], logistic))
    for i in 2:length(hids)
        push!(layer_list, Dense(hids[i-1], hids[i], logistic))
    end
    push!(layer_list, Dense(hids[end], output_size))
    push!(layer_list, softmax)

    # unpack arguments from layer_list and chain into mlp model
    return Chain(layer_list...)
end;

```

In [0]:

```

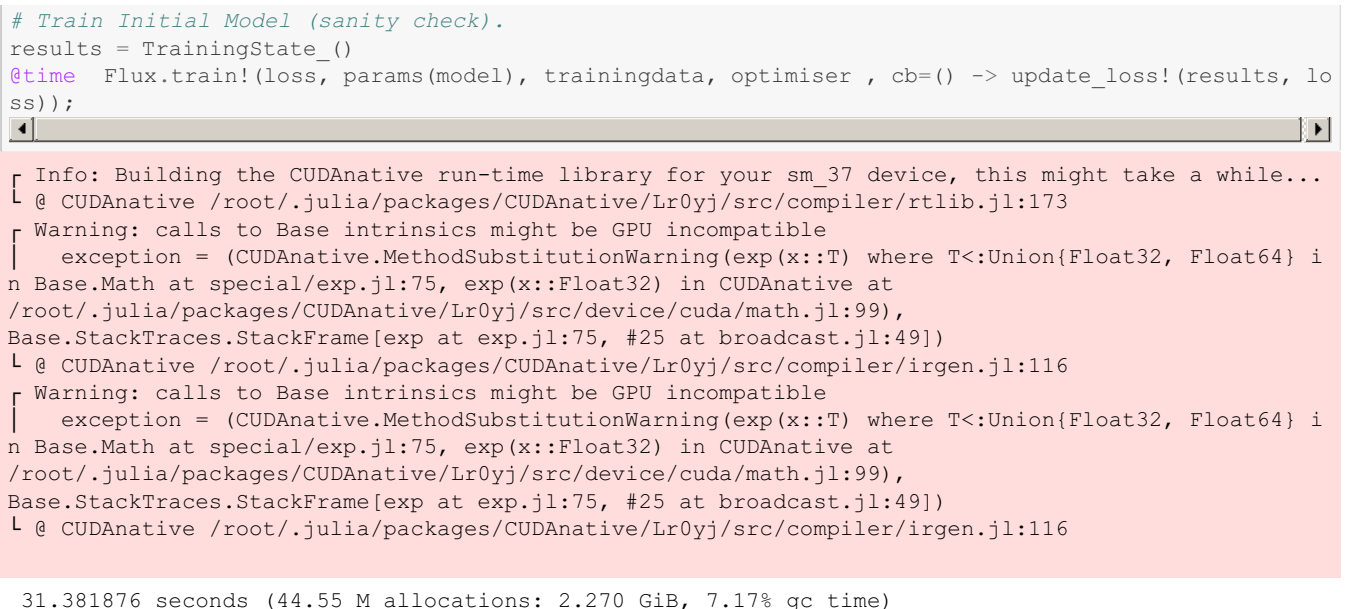
# Define a test model for sanity checking purposes
using Base.Iterators: repeated
using Flux: crossentropy

model = build_mlp(300, 5, [16]) |> gpu

# Define training environment
loss(x, y) = crossentropy(model(x), y)
epochs = 100
trainingdata = repeated((trainxs, trainys), epochs)
callback = () -> println("\$(loss(trainxs, trainys))\t$(loss(validxs, validys))")
optimiser = ADAM()

```

```
# Train Initial Model (sanity check).
results = TrainingState_()
@time Flux.train!(loss, params(model), trainingdata, optimiser, cb=() -> update_loss!(results, loss));
```



```
Info: Building the CUDAnative run-time library for your sm_37 device, this might take a while...
└─ @ CUDAnative /root/.julia/packages/CUDAnative/Lr0yj/src/compiler/rtlib.jl:173
Warning: calls to Base intrinsics might be GPU incompatible
└─ exception = (CUDAnative.MethodSubstitutionWarning(exp(x::T) where T<:Union{Float32, Float64} in Base.Math at special/exp.jl:75, exp(x::Float32) in CUDAnative at /root/.julia/packages/CUDAnative/Lr0yj/src/device/cuda/math.jl:99), Base.StackTraces.StackFrame[exp at exp.jl:75, #25 at broadcast.jl:49])
└─ @ CUDAnative /root/.julia/packages/CUDAnative/Lr0yj/src/compiler/irgen.jl:116
Warning: calls to Base intrinsics might be GPU incompatible
└─ exception = (CUDAnative.MethodSubstitutionWarning(exp(x::T) where T<:Union{Float32, Float64} in Base.Math at special/exp.jl:75, exp(x::Float32) in CUDAnative at /root/.julia/packages/CUDAnative/Lr0yj/src/device/cuda/math.jl:99), Base.StackTraces.StackFrame[exp at exp.jl:75, #25 at broadcast.jl:49])
└─ @ CUDAnative /root/.julia/packages/CUDAnative/Lr0yj/src/compiler/irgen.jl:116

31.381876 seconds (44.55 M allocations: 2.270 GiB, 7.17% gc time)
```

4.2. Explore model variants with different pooling operations, keeping the word embeddings fixed during training or updating them, different depths and layer sizes for the multi-layer perceptron, etc. to find what works well on the validation set and describe the process through which you arrive at your final model.

To find our preferred architecture and model parameters we used heuristics to explore different model architectures, and then fine tuned the model parameters on our chosen architecture. Once again, we aimed only to find a good model, that trains in a reasonable time, and did not try to over-optimize as we did not have the computational resources / time.

1. Heuristic Architecture Experimentation

To begin with, we focused on the embeddings and pooling operations and then moved onto considering the MLP.

Embeddings Experimentation

In our initial model, we processed the sentences by mapping them to the embeddings and applying the pooling operations to them prior to the training of the MLP, thus keeping the embeddings fixed while training. We then tried updating the embeddings during training by creating a function which maps each sentence to the embeddings and then applies the pooling operation(s) to it. We then inserted this function as the first layer of our MLP model. This process however did lead to overfitting and an unstable performance of the loss. It is also worth noting that adding an embedding and pooling layer into the model significantly increases the computation time. This observation agrees with the literature in sentiment analysis where, in many cases, fixed embeddings are chosen in favour to an embedding layer. Thus, we chose the fixed embeddings over updating embeddings during training. The function and model used for updating embeddings during training however, are given below.

Update embeddings during training

```
function one_sentence_processing(x)

    n = size(x)[1]
    matrix = zeros(300, n)
    i = 1
    for token in x
        matrix[:,i] = embedding(token)
        i += 1
    end

    # Mean & Max pool
    x_mean = zeros((300,1))
    x_max = zeros((300,1))

    for i in 1:300
        x_mean[i] = mean(matrix[i,:])
        x_max[i] = maximum(matrix[i,:])
    end
end
```



```

end

output = vcat(x_mean, x_max) |> gpu;

return matrix |> gpu
end

# Define model

model_2 = Chain(
  x -> one_sentence_processing(x),
  Dense(600, 128, logistic),
  Dense(128, 64, logistic),
  Dense(64, 5),
  softmax,
) |> gpu

```

Pooling Operations

Next we experimented with different pooling operations, that is maximum, mean and a combination of max & mean. Experimenting with these whilst also changing the architecture and the parameters of the model, we observed that in most cases max pooling led to worse results than mean pooling. This agrees with our intuition about the embedded space: that collecting information about the relative encoded "meaning" from every word of a sentence, aggregating that meaning, and placing it into its corresponding spot in this embedded space, is more likely to capture more nuanced information than simply deriving this information from the single most sentimentally dominant word. We also observed that the combination of mean and max pooling outputs were similar, but usually slightly worse results than mean pooling. Therefore, we chose mean pooling as the pooling operation for our final model.

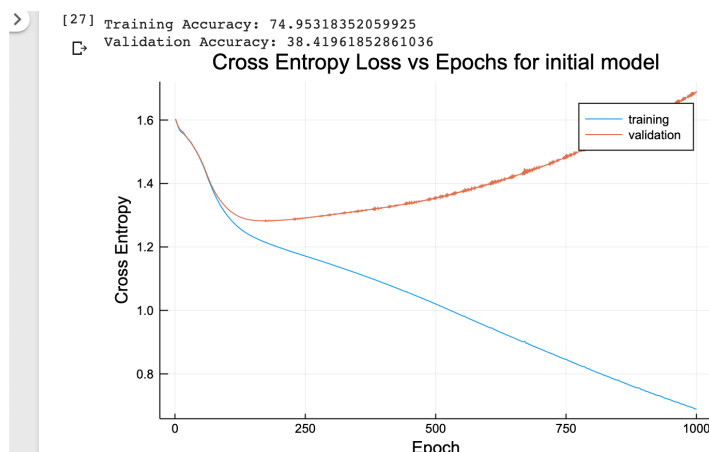
MLP Experimentation

In the case of an MLP the architecture parameters which can vary are: the number of hidden layers, the depths of the layers and the activation function. At this stage we kept the default values for hyperparameters such as the learning rate.

First we experimented with the number of hidden layers, h . We let $h = 2$ and increased them until the reduction in validation set loss was insignificant in comparison to the extra computation time. When using $h \geq 4$ we did not observe an increase in performance, but instead observed overfitting from a very early stage. Hence, we decided to explore the following range in the grid search: $h = 1, 2, 3$.

Experimenting with the number of neurons, N_h , again we performed a rough grid search on a one-layer network and we found that values below 8 or beyond 256 never led to a decrease in loss on the validation set. This led us to consider the range of $N_h = 8, 16, 32, 64, 128, 256$ in the grid search, in which the numbers of neurons are slightly smaller than in question 1 since the size of the data set in this question is smaller and different.

The three most commonly used activation functions include the logistic function, tanh and ReLu. We found that tanh and ReLu gave very high accuracy on the training set (around 75%), but overfit on the validation set from a very early stage and suffer from vanishing gradient problems, while the logistic function gave the best results on the validation set and did not have problems with vanishing gradient. An example of the accuracies and the loss plot using the ReLu activation function is shown below (notice the extreme overfitting from epoch 200 onwards):



Therefore, we decided to use the logistic activation function to avoid this problem.

It is also worth noting that many overfitting problems during the training of this model could have been due to the relatively small size of the data set which contains 8544 vectors (rather than matrices as in most of the other questions) of size 300x1.

Since we are dealing with a multi-class classification problem we decided to use softmax in the last layer as the softmax delivers output which can be interpreted as a probability distribution.

2. Rough Network Architecture Search

Using the above defined ranges, we ran a rough architecture grid search, considering all the possible permutations of the number of layers and the number of neurons in a similar way as in question 1. We include the results below.

We obtained the lowest validation loss when using $N_{h_1} = 256$, $N_{h_2} = 256$, $N_{h_3} = 64$ neurons in the hidden layers. This, however, is a fairly big model, which inevitably led to an increase in computational time. We also observed very good results for the model when $N_{h_1} = 128$, $N_{h_2} = 16$, $N_{h_3} = 0$, i.e using only 2 hidden layers. Since we are observing different architectures resulting in very similar losses it is worth checking a set of three possible models. Below we include our "exploratory" architecture search, as a code comment:

```
# we try all combinations of hidden layers up to 3 (4 including output)

hidden_units = [2^n for n in 3:8]

hidden_layers = []
# 1 layers
for h in hidden_units
    push!(hidden_layers, [h, 0, 0])
end;

# 2 layers
for h1 in hidden_units
    for h2 in hidden_units
        push!(hidden_layers, [h1, h2, 0])
    end
end;

# 3 layers
for h1 in hidden_units
    for h2 in hidden_units
        for h3 in hidden_units
            push!(hidden_layers, [h1, h2, h3])
        end
    end
end;

hidden_layers = reduce(hcat, hidden_layers)';
num_params = size(hidden_layers)[1];

using ProgressBars
using DataFrames: DataFrame
using Base.Iterators: repeated
using Flux: crossentropy

df_hids = DataFrame(hids1 = hidden_layers[:, 1], hids2 = hidden_layers[:, 2], hids3 = hidden_layers[:, 3], Train_Loss = zeros(num_params), Val_Loss=zeros(num_params))

for row in ProgressBar(eachrow(df_hids))

    # define model
    hids = [row.hids1, row.hids2, row.hids3]
    hids = filter(x -> x != 0, hids)

    model = build_mlp(300, 5, hids) |> gpu
```

```
# train
# define training environment
loss(x, y) = crossentropy(model(x), y)
epochs = 1000
trainingdata = repeated((trainxs, trainys), epochs)
callback = () -> println("${loss(trainxs, trainys)}\t${loss(validxs, validys)}")
optimiser = ADAM()
results = TrainingState_()

@time Flux.train!(loss, params(model), trainingdata, optimiser, cb=() -> update_loss!(results, loss));

row.Train_Loss = minimum(results.train_losses)
row.Val_Loss = minimum(results.valid_losses)
end;
```

Best three results from the grid search:

hids1	hids2	hids3	Train Loss	Val Loss
256	256	64	1.19565773	1.286618352
256	128	32	1.192667603	1.2886374
128	16	0	1.206823707	1.289077163

Extract Best Model

```
best_row_ix = argmin(df_hids[:, :Val_Loss])
best_row = df_hids[best_row_ix, :]
best_hids = [best_row.hids1, best_row.hids2, best_row.hids3];
```

3. Select subset of architectures

After analysing the results of the grid search, we decided to select a varied subset of architectures to proceed further with:

- $N_{h_1} = 256, N_{h_2} = 256, N_{h_3} = 64$, as this model gave the lowest validation loss on the grid search. - $N_{h_1} = 128, N_{h_2} = 16, N_{h_3} = 0$, as it gave the third best loss, but only uses 2 hidden layers.
- $N_{h_1} = 256, N_{h_2} = 128, N_{h_3} = 32$, as it gave the second small loss.

In [0]:

```
# Define subset of models:
hids_vars = [[256, 256, 64], [128, 16, 0], [256, 128, 32]];
```

4. Heuristic Hyperparameter Experimentation

Using the subset of models, we explored different optimisers and different learning rates.

We tested out the following optimisers: Adam (which we have used so far as default), SGD and Momentum and heuristically altered the learning rate. We trained the models to convergence (~1000 epochs) and compared validation accuracies.

In [0]:

```
# Pick one of the pre-selected models
chosen_hids = hids_vars[1]

# Define model
chosen_hids = filter(x -> x != 0, chosen_hids)
experimental_model = build_mlp(300, 5, chosen_hids) |> gpu

# train
```

```

loss(x, y) = crossentropy(experimental_model(x), y)
epochs = 1000
trainingdata = repeated((trainxs, trainys), epochs)
callback = () -> println("\$(loss(trainxs, trainys))\t$(loss(validxs, validys))")
optimiser = Momentum(0.01)
results = TrainingState_()

@time Flux.train!(loss, params(experimental_model), trainingdata, optimiser, cb=() -> update_loss!(results, loss));

```

19.747096 seconds (11.10 M allocations: 514.474 MiB, 1.53% gc time)

In [0]:

```

println("Accuracy: $(accuracy(sstvalidys, max_p(experimental_model(validxs)))) % on experimental model: $(chosen_hids)")

```

```

└ Warning: Performing scalar operations on GPU arrays: This is very slow, consider disallowing the se operations with `allowscalar(false)`
└ @ GPUArrays /root/.julia/packages/GPUArrays/tIMl5/src/indexing.jl:16

```

Accuracy: 25.340599455040874% on experimental model: [256, 256, 64]

Using this method, we consistently found that Adam significantly outperformed both SGD and Momentum. Further experimentation with learning rates in Adam, we found that the default learning rate of ADAM (0.001) delivered the best results on the validation set.

5. Select best model from the subset of models

Using the best hyperparameters across the three models, we then trained all of our selected models to convergence and picked the one which gives the best validation accuracy.

In [0]:

```

hids_vars_ = reduce(hcat, hids_vars)';
experimental_params = size(hids_vars)[1];

```

In [0]:

```

df_model = DataFrame(hids1 = hids_vars[:, 1], hids2 = hids_vars[:, 2], hids3 = hids_vars[:, 3],
Train_Loss = zeros(experimental_params), Val_Loss=zeros(experimental_params), Val_Acc = zeros(exper
imental_params));

```

In [0]:

```

for row in ProgressBar(eachrow(df_model))

    # define model
    hids = [row.hids1, row.hids2, row.hids3]
    hids = filter(x -> x != 0, hids)
    model = build_mlp(300, 5, hids) |> gpu

    # train

    loss(x, y) = crossentropy(model(x), y)
    epochs = 1000
    trainingdata = repeated((trainxs, trainys), epochs)
    callback = () -> println("\$(loss(trainxs, trainys))\t$(loss(validxs, validys))")
    optimiser = ADAM()
    results = TrainingState_()

    @time Flux.train!(loss, params(model), trainingdata, optimiser, cb=() -> update_loss!(results, loss));

    row.Train_Loss = minimum(results.train_losses)
    row.Val_Loss = minimum(results.valid_losses)
    row.Val_Acc = accuracy(sstvalidys, max_p(model(validxs)))
end;

```

```
18.885855 seconds (10.31 M allocations: 469.563 MiB, 1.36% gc time)
0.00% | 0/3 Inf:Inf<Inf:Inf, 0.00 it/
27.232457 seconds (7.81 M allocations: 362.766 MiB, 5.81% gc time)
33.33% | 1/3 00:47<Inf:Inf, 0.00 it/
] 30.662066 seconds (9.91 M allocations: 451.317 MiB, 5.26% gc time)
100.00% | 3/3 01:18<00:00, 0.03 it
/s]
```

In [0]:

```
df_model
```

Out[0]:

3 rows × 6 columns

	hids1	hids2	hids3	Train_Loss	Val_Loss	Val_Acc
	Int64	Int64	Int64	Float64	Float64	Float64
1	256	256	64	1.21377	1.29716	42.6885
2	128	16	0	1.26525	1.31063	42.0527
3	256	128	32	1.20435	1.29226	43.1426

We define our best model as the one which givest highest validation accuracy.

In [0]:

```
best_row_ix = argmax(df_model[:, :Val_Acc])
best_row = df_model[best_row_ix, :]
best_hids = [best_row.hids1, best_row.hids2, best_row.hids3];
```

Finally, we experimented again on our final model by using different pooling operations to make sure that our initial assumptions were right, and we concluded that average pooling gave the best results on the validation set.

4.3 Train your final model to convergence on the training set using an optimisation algorithm of your choice.

Training the best model, $N_{h_1} = 128$, $N_{h_2} = 16$, to convergence.

In [0]:

```
# Define model
best_hids = filter(x -> x != 0, best_hids)
final_model = build_mlp(300, 5, best_hids) |> gpu

# train

loss(x, y) = crossentropy(final_model(x), y)
epochs = 1200
trainingdata = repeated((trainxs, trainys), epochs)
callback = () -> println("$ (loss(trainxs, trainys)) \t $(loss(validxs, validys))")
optimiser = ADAM(0.001)
results = TrainingState_()

@time Flux.train!(loss, params(final_model), trainingdata, optimiser, cb=() -> update_loss!(results, loss));
```

```
17.603106 seconds (12.29 M allocations: 560.531 MiB, 1.85% gc time)
```

4.4 Provide a plot of the loss on the training set and validation set for each epoch of training.

In [0]:

```
plt = plot(1:epochs, results.train_losses, title="Cross Entropy Loss vs Epochs for final model", label = "training")
plot(1:epochs, results.valid_losses, label = "validation")
xlabel! ("Epoch")
ylabel! ("Cross Entropy")
plot(plt)
```

Out[0]:

4.5 Provide the final accuracy on the training, validation, and test set.

In [0]:

```
println("Training Accuracy: $(accuracy(ssttrainys, max_p(final_model(trainxs))))")
println("Validation Accuracy: $(accuracy(sstvalidys, max_p(final_model(validxs))))")
println("Test Accuracy: $(accuracy(ssttestys, max_p(final_model(testxs))))")
```

Training Accuracy: 48.37312734082397
 Validation Accuracy: 41.507720254314265
 Test Accuracy: 44.34389140271493

4.6 Provide a selection of the classification decisions of the final model for 5 opinionated sentences from movies reviews that you find online – perhaps for a movie you liked, or did not like? Remember to provide a links to the reviews from which you selected the sentences.

In [0]:

```
s1 = ["The", "film", "as", "a", "whole", "is", "very", "funny"]
s2 = ["It's", "an", "admirable", "move", "casting", "aside", "standard", "war", "movie", "clichés", "in", "favour", "of", "gritty", "realism"]
s3 = ["Among", "the", "other", "aspects", "of", "this", "unfunny", "and", "heartless", "Hogmanay", "Horror", "is", "Jessica", "Biel"]
s4 = ["I", "happen", "to", "dislike", "the", "film", "as", "heartily", "as", "anything", "I've", "seen", "in", "the", "past", "decade"]
s5 = ["she", "doesn't", "invest", "the", "character", "with", "style", "or", "with", "substance"]

sentences = sentence_processing([s1, s2, s3, s4,s5]) |> gpu

max_p(model(sentences))
```

Out[0]:

```
5-element Array{Int64,1}:
 3
 3
 0
 1
 1
```

Links to the movie reviews:

https://www.theguardian.com/film/News_Story/Critic_Review/Guardian_Film_of_the_week/0.4267.1015099.00.html \

<https://www.empireonline.com/movies/reviews/black-hawk-review/> \

<https://www.theguardian.com/film/2011/dec/08/new-years-eve-film-review> \

<https://www.newyorker.com/magazine/2019/10/07/todd-phillips-joker-is-no-laughing-matter> \

<https://www.newyorker.com/culture/richard-brody/the-great-gatsby-try-again-old-sport>

In [0]:

```
%%shell
if ! command -v julia 2>&1 > /dev/null
then
    wget 'https://julialang-s3.julialang.org/bin/linux/x64/1.0/julia-1.0.5-linux-x86_64.tar.gz' \
        -O /tmp/julia.tar.gz
    tar -x -f /tmp/julia.tar.gz -C /usr/local --strip-components 1
    rm /tmp/julia.tar.gz
fi
julia -e "using Pkg; pkg\"add IJulia; precompile;\""

--2019-12-13 09:42:20-- https://julialang-s3.julialang.org/bin/linux/x64/1.0/julia-1.0.5-linux-x86_64.tar.gz
Resolving julialang-s3.julialang.org (julialang-s3.julialang.org)... 151.101.2.49, 151.101.66.49, 151.101.130.49, ...
Connecting to julialang-s3.julialang.org (julialang-s3.julialang.org)|151.101.2.49|:443...
connected.
HTTP request sent, awaiting response... 302 gce internal redirect trigger
Location: https://storage.googleapis.com/julialang2/bin/linux/x64/1.0/julia-1.0.5-linux-x86_64.tar.gz [following]
--2019-12-13 09:42:20-- https://storage.googleapis.com/julialang2/bin/linux/x64/1.0/julia-1.0.5-linux-x86_64.tar.gz
Resolving storage.googleapis.com (storage.googleapis.com)... 64.233.167.128, 2a00:1450:400c:c04::80
Connecting to storage.googleapis.com (storage.googleapis.com)|64.233.167.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 88706549 (85M) [application/octet-stream]
Saving to: '/tmp/julia.tar.gz'

/tmp/julia.tar.gz  100%[=====>]  84.60M  50.4MB/s   in 1.7s

2019-12-13 09:42:23 (50.4 MB/s) - '/tmp/julia.tar.gz' saved [88706549/88706549]

Cloning default registries into /root/.julia/registries
Cloning registry General from "https://github.com/JuliaRegistries/General.git"
Resolving package versions...
Installed Conda _____ v1.3.0
Installed ZMQ _____ v1.0.0
Installed VersionParsing ____ v1.2.0
Installed Parsers _____ v0.3.10
Installed IJulia _____ v1.20.2
Installed BinaryProvider ____ v0.5.8
Installed MbedTLS _____ v0.6.8
Installed JSON _____ v0.21.0
Installed SoftGlobalScope - v1.0.10
Updating `~/julia/environments/v1.0/Project.toml`
[7073ff75] + IJulia v1.20.2
Updating `~/julia/environments/v1.0/Manifest.toml`
[b99e7846] + BinaryProvider v0.5.8
[8f4d0f93] + Conda v1.3.0
[7073ff75] + IJulia v1.20.2
[682c06a0] + JSON v0.21.0
[739be429] + MbedTLS v0.6.8
[69de0a69] + Parsers v0.3.10
[b85f4697] + SoftGlobalScope v1.0.10
[81def892] + VersionParsing v1.2.0
[c2297ded] + ZMQ v1.0.0
[2a0f44e3] + Base64
[ade2ca70] + Dates
[8ba89e20] + Distributed
[7b1f6079] + FileWatching
[b77e0a4c] + InteractiveUtils
[76f85450] + LibGit2
[8f399da3] + Libdl
[37e2e46d] + LinearAlgebra
[56ddb016] + Logging
[d6f4376e] + Markdown
[a63ad114] + Mmap
[44cfe95a] + Pkg
[de0858da] + Printf
[3fa0cd96] + REPL
[9a3f8284] + Random
[8c564d8e] + SHA
```

```
[e80e919c] + SHA
[9e88b42a] + Serialization
[6462fe0b] + Sockets
[8dfed614] + Test
[cf7118a7] + UUIDs
[4ec0a83e] + Unicode
Building Conda → `~/.julia/packages/Conda/kLXeC/deps/build.log`
Building ZMQ → `~/.julia/packages/ZMQ/ABGOx/deps/build.log`
Building MbedTLS → `~/.julia/packages/MbedTLS/X4xar/deps/build.log`
Building IJulia → `~/.julia/packages/IJulia/F1GUo/deps/build.log`
Precompiling project...
Precompiling IJulia
```

In [0]:

```
using Pkg

Pkg.add(Pkg.PackageSpec(;name="CuArrays", version=v"1.3.0"))
Pkg.add(Pkg.PackageSpec(;name="Flux", version=v"0.9.0"))

pkg"add Plots"
pkg"add StatsBase"
pkg"add Embeddings"
Pkg.add("ProgressBars")

pkg"precompile"

using CuArrays
using Flux
using Flux: onehot, chunk, batchseq, throttle, crossentropy
using StatsBase: wsample
using Embeddings

using Base.Iterators: partition
using LinearAlgebra
using Random
using Statistics
using Plots
using ProgressBars

Updating registry at `~/.julia/registries/General`
Updating git-repo `https://github.com/JuliaRegistries/General.git`
Resolving package versions...
Updating `~/.julia/environments/v1.0/Project.toml`
[no changes]
Updating `~/.julia/environments/v1.0/Manifest.toml`
[no changes]
Resolving package versions...
Updating `~/.julia/environments/v1.0/Project.toml`
[no changes]
Updating `~/.julia/environments/v1.0/Manifest.toml`
[no changes]
Resolving package versions...
Updating `~/.julia/environments/v1.0/Project.toml`
[no changes]
Updating `~/.julia/environments/v1.0/Manifest.toml`
[no changes]
Resolving package versions...
Updating `~/.julia/environments/v1.0/Project.toml`
[no changes]
Updating `~/.julia/environments/v1.0/Manifest.toml`
[no changes]
Resolving package versions...
Updating `~/.julia/environments/v1.0/Project.toml`
[no changes]
Updating `~/.julia/environments/v1.0/Manifest.toml`
[no changes]
Precompiling project...
```

Internal error: encountered unexpected error in runtime:


```

BoundsError(a=Array{Core.Compiler.BasicBlock, (32,)}[
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=1, last=6), preds=Array{Int64,
(1,)}[32], succs=Array{Int64, (1,)}[2]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=7, last=13), preds=Array{Int64,
(1,)}[1], succs=Array{Int64, (2,)}[5, 3]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=14, last=19), preds=Array{Int64,
(1,)}[2], succs=Array{Int64, (1,)}[4]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=20, last=20), preds=Array{Int64,
(1,)}[3], succs=Array{Int64, (1,)}[7]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=21, last=27), preds=Array{Int64,
(1,)}[2], succs=Array{Int64, (1,)}[6]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=28, last=28), preds=Array{Int64,
(1,)}[5], succs=Array{Int64, (1,)}[7]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=29, last=46), preds=Array{Int64,
(2,)}[4, 6], succs=Array{Int64, (2,)}[9, 8]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=47, last=48), preds=Array{Int64,
(1,)}[7], succs=Array{Int64, (0,)}[]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=49, last=49), preds=Array{Int64,
(1,)}[7], succs=Array{Int64, (1,)}[10]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=50, last=51), preds=Array{Int64,
(1,)}[9], succs=Array{Int64, (1,)}[11]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=52, last=52), preds=Array{Int64,
(1,)}[10], succs=Array{Int64, (1,)}[12]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=53, last=53), preds=Array{Int64,
(1,)}[11], succs=Array{Int64, (1,)}[13]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=54, last=55), preds=Array{Int64,
(1,)}[12], succs=Array{Int64, (1,)}[14]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=56, last=64), preds=Array{Int64,
(1,)}[13], succs=Array{Int64, (1,)}[15]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=65, last=69), preds=Array{Int64,
(1,)}[14], succs=Array{Int64, (2,)}[17, 16]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=70, last=72), preds=Array{Int64,
(1,)}[15], succs=Array{Int64, (0,)}[]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=73, last=80), preds=Array{Int64,
(1,)}[15], succs=Array{Int64, (2,)}[19, 18]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=81, last=82), preds=Array{Int64,
(1,)}[17], succs=Array{Int64, (0,)}[]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=83, last=83), preds=Array{Int64,
(1,)}[17], succs=Array{Int64, (1,)}[20]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=84, last=85), preds=Array{Int64,
(1,)}[19], succs=Array{Int64, (1,)}[21]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=86, last=86), preds=Array{Int64,
(1,)}[20], succs=Array{Int64, (1,)}[22]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=87, last=87), preds=Array{Int64,
(1,)}[21], succs=Array{Int64, (1,)}[23]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=88, last=88), preds=Array{Int64,
(1,)}[22], succs=Array{Int64, (1,)}[24]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=89, last=90), preds=Array{Int64,
(1,)}[23], succs=Array{Int64, (1,)}[25]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=91, last=91), preds=Array{Int64,
(1,)}[24], succs=Array{Int64, (1,)}[26]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=92, last=92), preds=Array{Int64,
(1,)}[25], succs=Array{Int64, (1,)}[27]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=93, last=93), preds=Array{Int64,
(1,)}[26], succs=Array{Int64, (2,)}[29, 28]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=94, last=96), preds=Array{Int64,
(1,)}[27], succs=Array{Int64, (0,)}[]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=97, last=100), preds=Array{Int64,
(1,)}[27], succs=Array{Int64, (2,)}[31, 30]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=101, last=101), preds=Array{Int64,
(1,)}[29], succs=Array{Int64, (1,)}[32]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=102, last=102), preds=Array{Int64,
(1,)}[29], succs=Array{Int64, (1,)}[32]),
  Core.Compiler.BasicBlock(stmts=Core.Compiler.StmtRange(first=103, last=105), preds=Array{Int64,
(2,)}[30, 31], succs=Array{Int64, (1,)}[1]), i=(0,))
rec_backtrace at /buildworker/worker/package_linux64/build/src/stackwalk.c:94
record_backtrace at /buildworker/worker/package_linux64/build/src/task.c:246
jl_throw at /buildworker/worker/package_linux64/build/src/task.c:577
jl_bounds_error_ints at /buildworker/worker/package_linux64/build/src/rtutils.c:187
getindex at ./array.jl:731
jfptr_getindex_2271.clone_1 at /usr/local/lib/julia/sys.so (unknown line)
jl_apply_generic at /buildworker/worker/package_linux64/build/src/gf.c:2162
replace_code_newstyle! at ./compiler/ssair/legacy.jl:86
optimize at ./compiler/optimize.jl:212
typeinf at ./compiler/typeinfer.jl:35
typeinf_ext at ./compiler/typeinfer.jl:574
typeinf_ext at ./compiler/typeinfer.jl:611

```

```

jfp_ptr_typeinfo_ext_1.clone_1 at /usr/local/lib/julia/sys.so (unknown line)
jl_apply_generic at /buildworker/worker/package_linux64/build/src/gf.c:2162
jl_apply at /buildworker/worker/package_linux64/build/src/julia.h:1537 [inlined]
jl_apply_with_saved_exception_state at /buildworker/worker/package_linux64/build/src/rtutils.c:257
jl_type_infer at /buildworker/worker/package_linux64/build/src/gf.c:253
jl_compile_method_internal at /buildworker/worker/package_linux64/build/src/gf.c:1764 [inlined]
jl_fptr_trampoline at /buildworker/worker/package_linux64/build/src/gf.c:1808
jl_apply_generic at /buildworker/worker/package_linux64/build/src/gf.c:2162
jl_apply at /buildworker/worker/package_linux64/build/src/julia.h:1537 [inlined]
start_task at /buildworker/worker/package_linux64/build/src/task.c:268
unknown function (ip: 0xffffffffffffffff)

```

Assignment 2: “Half-bag”

2.5 Sequence encoding

Data

Stanford Sentiment Treebank

In [0]:

```

run(`curl -fsS https://nlp.stanford.edu/sentiment/trainDevTestTrees_PTB.zip -o
/tmp/trainDevTestTrees_PTB.zip`)
run(`unzip -q -o -d /tmp /tmp/trainDevTestTrees_PTB.zip`)
run(`rm -f /tmp/trainDevTestTrees_PTB.zip`)

nothing

```

In [0]:

```

function loadsst(path)
    xs = Array{String}[]
    ys = Int[]
    open(path) do file
        # Quick, dirty, and improper S-expression parsing.
        for line in eachline(file)
            soup = split(line)
            push!(ys, parse{Int, lstrip(first(soup), '(')})
            tokens = String[]
            for chunk in soup[2:end]
                endswith(chunk, ")") || continue
                push!(tokens, rstrip(chunk, '('))
            end
            push!(xs, tokens)
        end
    end
    xs, ys
end

ssttrainxs, ssttrainys = loadsst("/tmp/trees/train.txt")
sstvalidxs, sstvalidys = loadsst("/tmp/trees/dev.txt")
ssttestxs, ssttestys = loadsst("/tmp/trees/test.txt")

nothing

```

TrainingState struct to used in callbacks to record training state:

In [0]:

```

mutable struct TrainingState
    train_losses::Vector{Float64}
    valid_losses::Vector{Float64}
end
TrainingState() = TrainingState{[], []}

function update_loss!(state::TrainingState, loss)

```

```

# we already called Tracker.data
training_loss, validation_loss = loss()
push!(state.train_losses, training_loss)
push!(state.valid_losses, validation_loss)
end;

```

Util functions:

In [0]:

```
accuracy(y, y_preds) = 100*(mean(y .== y_preds));
```

In [0]:

```

function max_p(distr)
    preds = Vector{Int64}()
    for i in 1:size(distr)[2]
        push!(preds, argmax(distr[:, i]))
    end
    return preds .- 1;
end;

```

Embeddings:

Let's define a function that gives us embeddings for tokens

In [0]:

```

const w2v = load_embeddings(Word2Vec)
const token2index = Dict{token => i for (i, token) in enumerate(w2v.vocab)}
function embedding(token)
    # If a token is not in our vocabulary, we use "UNK" to represent it.
    token = token in w2v.vocab ? token : "UNK"
    w2v.embeddings[:, token2index[token]]
end

```

```

tcmalloc: large alloc 3600007168 bytes == 0xdcc8000 @ 0x7f7019ee1b6b 0x7f7019f01379
0x7f70197f45ec 0x7f70197be42b 0x7f6ff8848177 0x7f6ff884710c 0x7f701979d05c 0x7f70197a1ca7
0x7f6ff8c64be8 0x7f6ff8c64940 0x7f6ff8c64a28 0x7f70197a1ca7 0x7f6ff8c5fc99 0x7f6ff8c5fd5b
0x7f70197a1ca7 0x7f6ff8c5fa3a 0x7f70197a10d6 0x7f701990af10 0x7f701990ac39 0x7f701990b5bc
0x7f701990bd3f 0x7f70197b796c 0x7f701990c80d 0x7f70197d6ffc 0x7f70197b28e0 0x7f6ff8c1e22f
0x7f6ff8c1b87d 0x7f70197a10d6 0x7f70197af846 0x7f70197afea2 0x7f70076c0359

```

Out [0]:

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

Processing of data:

In [0]:

```

# takes an array of arrays of tokens corresponding to sentences, and returns
# an array with the same structure, where tokens have been replaced with embeddings, as defined by

# embed_func (this could be a neural embedding, or even just a onehot encoder)
# Note: DO NOT PAD BEFORE USING THIS FUNCTION - It handles padding via the character you pass it
# and you'll just slow it down!

function embed_sentences(sentences, embed_func, target_length; pad_token="_")
    pad_embedding = embed_func(pad_token)
    embedded_sentences = Array{Array{Array{Float32, 1}, 1}, 1}()

    for sentence in Progressbar(sentences)
        embedded_sentence = Array{Array{Float32, 1}, 1}()
        for token in sentence
            push!(embedded_sentence, embed_func(token))
        end
        n_remainder = target_length - length(sentence)
        append!(embedded_sentence, Iterators.repeated(pad_embedding, n_remainder))
        push!(embedded_sentences, embedded_sentence)
    end
end

```

```
end
return embedded_sentences
end;
```

Prepare and Batch Data

In [0]:

```

### PREP AND BATCH TRAINING DATA ###
training_seq_length = training_seq_length = maximum([maximum(map(length, ssttrainxs)), maximum(map(
length, sstvalidxs))])
batch_size = Int(round(length(ssttrainxs)/16))

embedded_sentences = embed_sentences(ssttrainxs, embedding, training_seq_length)
sequence_batches = collect(partition(embedded_sentences, batch_size))
XS_TRAIN = map(batchseq, sequence_batches);

##### DEFINE YS CORRESPONDING TO WORDS IN EACH SEQUENCE #####

onehot_ratings = []
unq_ys = sort!(unique(ssttrainys))

for rating in ssttrainys
    push!(onehot_ratings, collect(Iterators.repeated(onehot(rating, unq_ys), training_seq_length)))
end;

sequence_batches_YS = collect(partition(onehot_ratings, batch_size))
YS_TRAIN = map(batchseq, sequence_batches_YS);

```

```
77.00% ██████████ | 6579/8544 02:19<00:42, 47.29 it/
```

Excessive output truncated after 524373 bytes.

```
77.05% ██████████ | 6583/8544 02:19<00:41, 47.30 it/
```

s]

In [0]:

```

### PREP VALIDATION DATA ###
# don't batch validation data
batch_size = length(sstvalidxs)
embedded_sentences_val = embed_sentences(sstvalidxs, embedding, training_seq_length)
sequence_batches_val = collect(partition(embedded_sentences_val, batch_size))
XS_VAL = map(batchseq, sequence_batches_val);

##### DEFINE YS CORRESPONDING TO WORDS IN EACH SEQUENCE #####

onehot_ratings_val = []

for rating in sstvalidys
  push!(onehot_ratings_val, collect(Iterators.repeated(onehot(rating, unq_ys), training_seq_length))
)
end;

sequence_batches_YS_val = collect(partition(onehot_ratings_val, batch_size))
YS_VAL = map(batchseq, sequence_batches_YS_val);

```

[illegible]

1. Implement a recurrent neural network model which encodes a sequence of words into inputs to a multi-layer perceptron with a cross-entropy loss for the Stanford Sentiment Treebank data.

In [0]:

```
# Specify an initial model to ensure it works
```


In this question, we again refer to primarily manual experimentation with the use of "targeted" (not dense but smart) and "exploratory" (sparse but big) searches where appropriate.

Similar to Question 3, in this question the sequential delivery of data to our network is important, since we aim to further capture the structure and sentiment of a sentence by making our classifier "order-aware".

Sequential Batches (Data Input Structure)

We repeat our motivations from question 3, though this time we modify them for the sentiment analysis problem domain:

We are given a dataset of sentences corresponding to movie reviews, and we would like to create a model that learns to predict the next characters in a given sentence that comes from this distribution. This means that it is important to consider each sentence as an i.i.d sample. It was somewhat easier to initially get the model working with chunked together text, as we were able to simply lift and shift the code from the character sequencer for a shakespeare script in lectures. However, this did not match our problem domain, since concatenating and mixing movie reviews violates our i.i.d. assumption and perturbs the target distribution. It would chop together movie review sentences in a way that would very likely (and empirically did) hinder our model's learning.

Instead, we treat each sentence as a datapoint, and force them to be the same size so that we can make use of batching to train our models more efficiently. Batches consist of a collection of matrices, the first matrix in a batch corresponds to the first character of sentences in that batch, the second matrix corresponds to the second character of all the sentences in that batch, etc. We can then feed batches in a sequential manner, correctly training the RNN with respect to its hidden state, (as opposed to delivering each token from each sentence sequentially, one at a time).

The main problem this introduces that we do not experience with the 'text chunking' method, is that we must make our sequences all the same size (so that we can group them into batches). We do this via padding (which constitutes the majority of our preprocessing step!):

Padding

We again replicate our strategy for padding from Q3. Rerunning all of the discussed experiments that follow with the different forms of padding of sequences. We attempted to use post and pre-padding, where we pad to both the mean and maximum length in the training data. We found that max-post padding gave the best final results on our training data. We hypothesise that the reason that max padding worked in this question, (whereas it performed poorly in Q3), is due to the fact that the variance in the number of words (i.e. our tokens in this problem) in the sentence-reviews is significantly smaller than the variance in their character length. Hence, we do not triple our dataset in size by padding to maximum length, as we did in question 3. Instead the full benefit of sequential batching of our data is achieved, whilst maintaining all of the data in our dataset (since we don't have to truncate), and also whilst adding a relatively small amount of noise (though still quite significant!)

For computational efficiency, our implementation of padding is this time done in the embedded space, i.e. we add the padded characters AFTER we map sentences into embedded space, to avoid having to embed a large number of null characters. This can be seen above in the function `embed_sentences`.

Embeddings Experimentation

We experimented with fixing the embeddings versus including them as a trainable component of the model and allowing their weights to vary.

Fixed Embeddings

When fixing embeddings, we elected to process the sentences by mapping them to the embeddings prior to training the RNN and the MLP, thus keeping the embeddings fixed while training, but also allowing for a significant speed-up (since the embeddings both did not have to be trained, nor did they have to be computed for each data sample).

Trainable Embeddings

Perhaps as expected from our previous results in Q4, we saw no significant improvement when inserting the embeddings into the model and allowing them to update. This process led to overfitting of the data and an unstable performance on the loss. We suspect that in this version of the problem, we simply do not have enough data to be able to convert the heavily-pretrained embeddings to become specialised to our problem domain. It is also worth noting that adding the embedding layer into the model significantly increases the computation time. This observation agrees with the literature in sentiment analysis where in many cases fixed general embeddings are chosen in favour to a trainable embedding layer.

Thus, we chose the fixed embeddings over updating embeddings during training.

Recurrent Units Experimentation

Similar to Q3, we proceed by attempting to find a good architecture for the RNN which generally works well for any given "good" MLP, hence allowing us to assume a reasonable amount of independence between this component and the choice of architecture for the

final classification layers.*

-* Please refer to the original heuristic argument we give for our intuition here given in Q3. We do not repeat it here for brevity, but it's exactly the same idea.

Vanilla

We began by experimenting with the Vanilla RNN. Again, it vastly underperformed when compared to both GRU and LSTM units. Given our results in Q3, we would again expect this to be the case as this problem domain is arguably even more complex than in Q3. The Vanilla RNN performed significantly worse than the LSTM and the GRU units, which achieved both smaller losses and greater accuracies. Namely, using the Vanilla RNN, we did not observe a model which was able to achieve an accuracy of more than 28% on the validation set. We stopped investigating Vanilla architectures once the superiority of GRU and LSTM became clear.

GRU, LSTM

The key difference between a GRU and an LSTM is that a GRU has two gates (reset and update gates) whereas an LSTM has three gates (namely the input, output and forget gates), so the GRU controls the flow of information like the LSTM unit, but without having to use a (long-term) memory unit. In question 3 we found that the GRU networks tended to perform better, since the character-prediction task was very short term. In the case of sentiment analysis we care far more about longer-term dependencies, as a sentence's overall sentiment is encoded within a structured relationship spanning multiple words/tokens. Based on this, and the literature we could find on the topic, we expected LSTM to perform better than GRU, and this intuition was validated by observing significantly better validation loss and accuracy (on average) while using LSTM units.

We elected to use an LSTM unit with a single layer after our experiments showed that using 2 (as in Q3) did not significantly improve the trained models, and in fact made convergence take much longer.

MLP Experimentation

Again, once we fixed our RNN to be relatively robust to our choice of MLP, we were able to vary our MLP in a relatively independent manner.

The architectural parameters which vary are: the number of hidden layers, the sizes of the layers, and the choice of activation functions. At this stage we keep the default values for hyperparameters such as the learning rate.

In this question, training times were significantly higher than in question 4. This limited our ability to perform architectural grid searches. However, since we had set up our experiments such that we were able to assume a level of independence between the architecture of the RNN, (and the time-aware representation it constructs), and the architecture of the MLP, we were able to use our results from question 4 to motivate architectures for this question. We hypothesised that this would work due to their shared fundamental problem domain (the only difference being in the order-aware vs order-naive representations). This worked surprisingly well, with architectures that performed well in question 4 again performing relatively well in question 5.

Hence, based on our findings that a two hidden layer perceptron achieves satisfactory results in question 4 and given that our training times made a grid search impractical, we decided to manually experiment with MLPs with only 1 and 2 hidden layers. Experimenting with the number of neurons, we observed that relatively simple and small architectures (with number of neurons varying from 16 to 128) lead to better results than more complex and difficult to train architectures (256+ neurons). Trying a lot of different combinations we ended up achieving the best results with two hidden layers of size 128 and 64 neurons, respectively. This is the architecture that we end up training for our final implementation below.

We also tried a number of different activation functions in the models which performed well. We chose to try the three most commonly used activation functions within our dense layers, including the logistic function, tanh and ReLu. We find that the differences in the validation accuracy are very small between the three activation functions, but the ReLu activation function gave the best results on the validation set.

We lastly experimented with different learning rates for the optimiser that worked best across our best models, namely ADAM. We obtained really good results when using the default parameter 0.001, and when we increased it up to 0.01 we observed more and more strange behaviour for our loss function which quickly got stuck hovering around 81.5 (validation). We were unable to verify (due to computational time) whether if we had trained for more epochs (i.e. 500+) whether this problem would be overcome.

Note: Since we are dealing with a multi-class classification problem we fixed the MLP to use softmax in the last layer throughout our experiments, as the softmax delivers output which can be interpreted as a probability distribution.

Batch size

We again adjusted the batch size throughout to find out a reasonable number of batches to deliver, which balances good results while keeping the training time reasonably low. We also had the additional constraint in this problem, due to the fact that our (i.e. 300xfloat32) embeddings take up a significant amount of RAM, and hence we occasionally experienced computational issues/crashing during training when using large batch sizes. We concluded that creating 8 batches made training efficient (as it was fairly close to full-batch), while also leading to good results on the validation set. Smaller batch sizes resulted in infeasibly slow

training times, and we were pragmatically unable to verify whether they converged to better validation scores.

3. Train your final model to convergence on the training set using an optimisation algorithm of your choice.

In [0]:

```
# specify LSTM model with

logistic(x) = 1/(1 .+ exp.(-x));

m = Chain(
    LSTM(300, 128),
    Dense(128, 64, relu),
    Dense(64, 5),
    softmax)

m = gpu(m)

function loss(xs, ys)
    l = sum(crossentropy.(m.(gpu.(xs)), gpu.(ys)))
    Flux.reset!(m)
    return l
end

# global loss
function global_loss()
    zum_train = 0
    for i in 1:size(XS_TRAIN)[1]
        zum_train += Tracker.data(loss(XS_TRAIN[i], YS_TRAIN[i]))
    end
    training_loss = zum_train/size(XS_TRAIN)[1]

    zum_val = 0
    for i in 1:size(XS_VAL)[1]
        zum_val += Tracker.data(loss(XS_VAL[i], YS_VAL[i]))
    end
    validation_loss = zum_val/size(XS_VAL)[1]

    return training_loss, validation_loss
end

opt = ADAM()
tx, ty = (XS_VAL[1], YS_VAL[1])
evalcb = () -> @show loss(tx, ty)

results = TrainingState()
epochs = 130
for epoch in ProgressBar(1:epochs)
    Flux.train!(loss, params(m), zip(XS_TRAIN, YS_TRAIN), opt)
    update_loss!(results, global_loss)
    #evalcb()
end;
```

100.00%  130/130 08:00<00:00, 0.27 it /s]

4. Provide a plot of the loss on the training set and validation set for each epoch of training.

In [0]:

```
plt = plot(results.train_losses, title="Cross Entropy Loss vs Epochs for final model", label =
"training")
plot!(results.valid_losses, label = "validation")
xlabel!("Epoch")
ylabel!("Average Per-Batch Cross Entropy")
plot(plt)
```

Out[0]:

5. Provide the final accuracy on the training, validation, and test set.

In [0]:

```
using Flux: onecold

### GET SEQUENTIAL PREDICTIONS ###

m = cpu(m)
predictions_val = []
for sentence in ProgressBar(sstvalidxs)
    for token in sentence[1:end-1]
        # we don't care about the prediction until the end
        m(embedding(token))
    end
    final_output = argmax(m(embedding(sentence[end]))) - 1
    push!(predictions_val, final_output)
    Flux.reset!(m)
end;

predictions_train = []
for sentence in ProgressBar(ssttrainxs)
    for token in sentence[1:end-1]
        # we don't care about the prediction until the end
        m(embedding(token))
    end
    final_output = argmax(m(embedding(sentence[end]))) - 1
    push!(predictions_train, final_output)
    Flux.reset!(m)
end;

predictions_test = []
for sentence in ProgressBar(ssttestxs)
    for token in sentence[1:end-1]
        # we don't care about the prediction until the end
        m(embedding(token))
    end
    final_output = argmax(m(embedding(sentence[end]))) - 1
    push!(predictions_test, final_output)
    Flux.reset!(m)
end;
```

[illegible]

Excessive output truncated after 524324 bytes.

42.23%

In [0]:

```
println("Training Accuracy: $(accuracy(predictions_train, ssttrainys))")
println("Validation Accuracy: $(accuracy(predictions_val, sstvalidys))")
println("Test Accuracy: $(accuracy(predictions_test, ssttestys))")
```

```
Training Accuracy: 51.56835205992509
Validation Accuracy: 43.596730245231605
Test Accuracy: 44.6606334841629
```

6. Provide a selection of the classification decisions of the final model for 5 opinionated sentences from movies reviews that you find online – perhaps for a movie you liked, or did not like? Remember to provide a links to the reviews

from which you selected the sentences.

In [0]:

```
using Flux: onecold

s1 = ["The", "film", "as", "a", "whole", "is", "very", "funny"]
s2 = ["It's", "an", "admirable", "move", "casting", "aside", "standard", "war", "movie", "clichés",
"in", "favour", "of", "gritty", "realism"]
s3 = ["Among", "the", "other", "aspects", "of", "this", "unfunny", "and", "heartless", "Hogmanay",
"Horror", "is", "Jessica", "Biel"]
s4 = ["I", "happen", "to", "dislike", "the", "film", "as", "heartily", "as", "anything", "I've", "s
een", "in", "the", "past", "decade"]
s5 = ["she", "doesn't", "invest", "the", "character", "with", "style", "or", "with", "substance"]

sentences = [s1, s2, s3, s4, s5]

### GET PREDICTIONS ###

m = cpu(m)
predictions = []
for sentence in ProgressBar(sentences)
    for token in sentence[1:end-1]
        # we don't care about the prediction until the end
        m(embedding(token))
    end
    final_output = argmax(m(embedding(sentence[end]))) - 1
    push!(predictions, final_output)
    Flux.reset!(m)
end;

predictions
```

100.00%  5/5 00:00<00:00, 47.82 it /s]

Out[0]:

```
5-element Array{Any,1}:
 3
 3
 0
 1
 1
```

References of movie reviews:

https://www.theguardian.com/film/News_Story/Critic_Review/Guardian_Film_of_the_week/0.4267.1015099.00.html \

<https://www.empireonline.com/movies/reviews/black-hawk-review/> \

<https://www.theguardian.com/film/2011/dec/08/new-years-eve-film-review> \

<https://www.newyorker.com/magazine/2019/10/07/todd-phillips-joker-is-no-laughing-matter> \

<https://www.newyorker.com/culture/richard-brody/the-great-gatsby-try-again-old-sport>