TLDR README bigger than a regular readme and less thorough than a research paper.

A fairly complete reference for what I was doing while I was learning makemore.py and for whoever really wants to get into the details. Mostly on makemore with a few sidequests thrown in. All with generous appendices.

This code is a fork of Andrej Karpathy's makemore, (https://github.com/karpathy/makemore) it's purpose is to :

1. Tune hyperparameters for makemore using a grid search.

2. Experiment with smaller (1/10X) and larger datasets (25X) than names.txt Also a 10M extra large dataset of quotes.

3. Search for performance due to initialization lottery by changing seed values.

4. Add early stopping, loss target, options to makemore. And tagging to help evaluate the performance during grid search.

   a. Add logging of tag, step #, train and validation loss to CSV file for plotting and using the file for search on lowest test loss for grid search via sort and using the hex tag to reference the hyperparameters used for training.

   b. Bump the amount of the dataset to be used for validation at 10% to 100000 instead of 1000. This will resut in validation set being 10% for larger datasets. This was done late and is not reflected in some of the example runs shown in this document

5. Added a maximum parameters option to makemore that will exit the code is the model size is above the amount entered. This option allows for a grid search for the best hyperparameters with a constraint on model size. Without this larger models with more layers and larger embedding values always win the grid search.

The initial intention from the start was, to try to makemore on some different datasets other than names.txt.

Modifications

```
usage: makemore.py [-h] [--input-file INPUT_FILE] [--work-dir WORK_DIR]
                   [--resume] [--sample-only] [--num-workers NUM_WORKERS]
                   [--max-steps MAX_STEPS] [--device DEVICE] [--seed SEED]
                   [--top-k TOP_K] [--type TYPE] [--n-layer N_LAYER]
                   [--n-head N_HEAD] [--n-embd N_EMBD] [--n-embd2 N_EMBD2]
                   [--batch-size BATCH_SIZE] [--learning-rate LEARNING_RATE]
                   [--weight-decay WEIGHT_DECAY] [--lt-thr LT_THR]
                   [--early-stop EARLY_STOP] [--max-params MAX_PARAMS]

Make More

optional arguments:
  -h, --help            show this help message and exit
  --input-file INPUT_FILE, -i INPUT_FILE
                        input file with things one per line
  --work-dir WORK_DIR, -o WORK_DIR
                        output working directory
  --resume              when this flag is used, we will resume optimization
                        from existing model in the workdir
  --sample-only         just sample from the model and quit, don't train
  --num-workers NUM_WORKERS, -n NUM_WORKERS
                        number of data workers for both train/test
  --max-steps MAX_STEPS
                        max number of optimization steps to run for, or -1 for
```

```
                    infinite.
--device DEVICE     device to use for compute, examples:
                    cpu|cuda|cuda:2|mps
--seed SEED         seed
--top-k TOP_K       top-k for sampling, -1 means no top-k
--type TYPE         model class type to use,
                    bigram|mlp|rnn|gru|bow|transformer
--n-layer N_LAYER   number of layers
--n-head N_HEAD     number of heads (in a transformer)
--n-embd N_EMBD     number of feature channels in the model
--n-embd2 N_EMBD2   number of feature channels elsewhere in the model
--batch-size BATCH_SIZE, -b BATCH_SIZE
                    batch size during optimization
--learning-rate LEARNING_RATE, -l LEARNING_RATE
                    learning rate
--weight-decay WEIGHT_DECAY, -w WEIGHT_DECAY
                    weight decay
--lt-thr LT_THR, -t LT_THR
                    Loss target, stop when loss reaches the input value
                    target threshold.
--early-stop EARLY_STOP, -e EARLY_STOP
                    early stopping filter
--max-params MAX_PARAMS, -m MAX_PARAMS
                    Exits if parameters of the model are larger than
                    --max-params
```

Early Stopping

The modifications to makemore itself are an option for early stopping.
When the code runs and the loss stops to drop further, it looks at these 500 steps.
where the models normally save, looks to see if the loss has not decreased, then a watchdog count increases, and the commandline parameter for the early stopping is the filter of how many ties to wait for the loss to go down before quitting. So if it's one, then it goes through that iteration once, and decides to stop.
If 10 it goes through 10 times. That's change #1.

Loss Target

The other is a target threshold for loss.

When I was determining which batch size was best, I was looking to see how fast the code ran.So by changing batch size, makemore would go down to a loss quicker or slower depending on the batch size. So the idea is to run a script, have a target, and then see which batch sizes that are fed into the command line arguments for make more will produce the shortest run times. Change #2.

Maximum Parameters

This option allows specification of the maximum number of parameters for the makemore code to run. If the parameters of the model as calculated is greater than this optional amount the program exits. What this is for. When running a grid search for the best hyperparameters most of the time the larger models with more parameters will have the best loss. By constraining the size of the model it is possible to see what combination of layers, heads and learning rate result in best performance in the first 500 steps. Change #3.

Scripts

There are some scripts, one is for tuning hyperparameters.

It will vary the hyperparameters using a bash, shell script, and feed the parameters as

command line arguments to makemore.

With this in mind, the idea was not to disturb the makemore code too much by putting a test/tuning rig inside of it, but to use batch shell scripts to run it externally.

Tagging

There is a concept of tagging that goes along with this.
The idea is to tag specific outputs that makemore produces to stdout which could be instead piped to a file, including the dictionary of the hyperparameters gets tagged.
Whatever is the best loss that will get tagged, it steps with 500 and then saves the best loss if it is better than previous validation losses. This gets tagged and can be searched for later on in the output file when the batch file runs multiple iterations of makemore. If there's early stopping that gets tagged as well. What I mean by tag is a random hexadecimal number.
I chose 24 bits, figuring that the collision probability is fairly low, considering the length of runs that will be made.
The reason for this tagging is you can run the optimization script, either via nohup or run it, piping it into a file. So all the output is in some file. Now you can sort and look for the best loss. And when you get the best loss value at the bottom of your output, you can then look at the corresponding tag to tag for the early stopping best loss is at the end of it.
The tag for the loss is placed of the line so that it doesn't affect the sorting order of the loss values.
This allows the use of plain sort, you don't have to do anything else.
sort -r will sort and reverse order, lowest loss will be on the bottom.
Then you can look up by searching again using grep for the tag.
And if you find the tag that is next to the dictionary, you know what the hyperparameters were that gave that low loss. Change #4

Tagging example...

```
0xba7857 {'input_file': 'names.txt', 'work_dir': 'gridtest', 'resume': False,
'sample_only': False, 'num_workers': 4, 'max_steps': -1, 'device': 'cpu', 'seed':
3407, 'top_k': -1, 'type': 'transformer', 'n_layer': 2, 'n_head': 4, 'n_embd': 128,
'n_embd2': 32, 'batch_size': 64, 'learning_rate': 0.0005, 'weight_decay': 0.001,
'lt_thr': 10000000000.0, 'early_stop': 1}
1.9553791284561157 is best loss at step 6000, Early Stopping Applied. tag: 0xba7857
0x709c53 {'input_file': 'names.txt', 'work_dir': 'gridtest', 'resume': False,
'sample_only': False, 'num_workers': 4, 'max_steps': -1, 'device': 'cpu', 'seed':
3407, 'top_k': -1, 'type': 'transformer', 'n_layer': 4, 'n_head': 4, 'n_embd': 128,
'n_embd2': 32, 'batch_size': 64, 'learning_rate': 0.0005, 'weight_decay': 0.001,
'lt_thr': 10000000000.0, 'early_stop': 1}
1.9571641683578491 is best loss at step 6000, Early Stopping Applied. tag: 0x709c53
```

See Appendix 2 + 2A for a full example of optimization using tagging and model sizing.

Batch Size Test Example...

```
Batch Size optimization, batch size, timing using date.
16
```

```
Thu 17 Aug 2023 07:45:03 PM EDT
Thu 17 Aug 2023 07:45:45 PM EDT
----------------------
32
Thu 17 Aug 2023 07:45:45 PM EDT
Thu 17 Aug 2023 07:46:14 PM EDT
----------------------
64
Thu 17 Aug 2023 07:46:14 PM EDT
Thu 17 Aug 2023 07:46:57 PM EDT
----------------------
128
Thu 17 Aug 2023 07:46:57 PM EDT
Thu 17 Aug 2023 07:47:39 PM EDT
----------------------
256
Thu 17 Aug 2023 07:47:39 PM EDT
Thu 17 Aug 2023 07:50:01 PM EDT
----------------------
```

Initialization Lottery

Additionally, I tried an experiment with what I would call initialization lottery to see what happens if you change the seed. I generated a bunch of random seeds using a shell script to make multiple runs of makemore with different seed values.

Then using the tag looked for the one that stopped early with the best loss.

That was a quick and dirty experiment. I did see some variation in loss performance after let it iterate for a while. So it does seem that there are some luckier seed values than others sometimes. And all of this information is in here.

```
-------- Seed RUNS -----

#!/bin/bash

# Loop 100 times
for ((i = 1; i <= 1000; i++)); do
    random_number=$RANDOM
    #echo "Random number $i: $random_number"
    #python3 makemore.py -e 1 --seed $random_number
    python3 makemore.py --seed $random_number --max-steps 501

done
```

Runs of seed values that were the best and worst performers of a random 1100 draw of seeds for 500 iters.

Just by getting a decent seed it is somewhat possible to get better performance quickly.

```
python3 makemore.py --seed 19964 -e 10

500 - test loss 2.177150011062622 is the best so far, saving model to out/model.pt
```

```
tag: 0x3ae929
e1- 1.955086350440979 is best loss at step 12500, Early Stopping Applied. tag:
0x7102bf
e10 1.9526185989379883 is best loss at step 20500, Early Stopping Applied. tag:
0xa1b5c2

python3 makemore.py --seed 13153 -e 10

500- test loss 2.264131546020508 is the best so far, saving model to out/model.pt
tag: 0x20e75
e1 - 2.03120756149292 is best loss at step 10500, Early Stopping Applied. tag:
0x29faf0
e10 - 2.0251574516296387 is best loss at step 16500, Early Stopping Applied. tag:
0xe41361
Run to 20500...
step 20500 train loss: 1.7382861375808716 test loss: 2.0330066680908203
```

Smaller and Larger Datasets Tests

After watching Let's build GPT: from scratch, in code, spelled out. I noticed the exercises in the comments below the video. I seem to already have started on EX2. I had thought of the addition one previously as I have encountered this in 2018 in the project addition-rnn-example which references the paper http://arxiv.org/abs/1410.4615.
- EX2: Train the GPT on your own dataset of choice! What other data could be fun to blabber on about? (A fun advanced suggestion if you like: train a GPT to do addition of two numbers, i.e. a+b=c. You may find it helpful to predict the digits of c in reverse order, as the typical addition algorithm (that you're hoping it learns) would proceed right to left too. You may want to modify the data loader to simply serve random problems and skip the generation of train.bin, val.bin. You may want to mask out the loss at the input positions of a+b that just specify the problem using y=-1 in the targets (see CrossEntropyLoss ignore_index). Does your Transformer learn to add? Once you have this, swole doge project: build a calculator clone in GPT, for all of +-/. Not an easy problem. You may need Chain of Thought traces.) (From: Let's build GPT: from scratch, in code, spelled out.* https://www.youtube.com/watch?v=kCc8FmEb1nY)

The other experiment with makemore was to try different data on it. And there's some helper files as well, some like addition.py. It is a Python file, it generates examples of addition using numbers. This is a really restricted form because it basically doesn't have a plus operator or equals. I tried to just stick to the zero to nine numbers and not have any operators in it. A real minimum set.

```
Example:
number of unique characters in the vocabulary: 10
vocabulary:
0123456789
```

That way it might be easier for the transformer or whatever architecture to, suss out what

addition is without anything else that's informative. Plus it makes for less symbols the model has to deal with keeping it to the bare minimum. So what it does is that the addition Python code creates 10,000 examples of additions of numbers from one to a hundred. And then the output is to the right.

So you'd have, for example, if you're adding 11 to 33 and getting 44, the representation is 1, 1, 3, 3, 4, 4, and so on. It always has leading zeros, so this keeps the formatting consistent. The only inconsistency is when things add to a hundred, there's something like, 9901100, but I figured that's something worth seeing what happens in that case, I wanted to see if it would actually catch on to that. Interestingly, it does learn this addition example well. (Supposedly it is even easier for it if the answer is in reverse as I have seen in the addition-rnn-example. But, the transformer seemed smart enough to figure it out quickly enough in the standard format)

It also learns some of the examples that don't show up in the 10,000 within the addition.txt file. There's not every combination in the file. There's not like a 24 plus 42, is 66 and 42 plus 24 is 66, type situation for all of what it is trained on. It might have 42 plus 24 but the reverse is missing, and it does learn to generalize to the reverse case. If you watch it while it's training, you train it up to like 3500 steps, it's kind of learned enough, but it hasn't yet memorized all of it. So it still presents you with these unique examples that it's learning ones that it didn't see. That was a smaller data set. That's about 10 times smaller than the names. So it can overfit very quickly and memorize.

```
0x179281 {'input_file': 'addition.txt', 'work_dir': 'gridtest', 'resume': False,
'sample_only': False, 'num_workers': 4, 'max_steps': -1, 'device': 'cpu', 'seed':
3407, 'top_k': -1, 'type': 'transformer', 'n_layer': 4, 'n_head': 2, 'n_embd': 64,
'n_embd2': 32, 'batch_size': 32, 'learning_rate': 0.001, 'weight_decay': 0.001,
'lt_thr': 10000000000.0, 'early_stop': 1}
1.1848140954971313 is best loss at step 6000, Early Stopping Applied. tag: 0x179281
```

Note: I noticed that variations in 'n_embd2' did not make a difference in the best loss discovered via grid search. Looking at the code for the transformer this n_embd2 is not used. Which I learned after using the code for a bit.

Example of output:

```
step 2400 | loss 1.1747 | step time 19.74ms
-----------------------------------------------------------------------------
6 samples that are in train:
830689
821294
033437
8812100
064248
393372
0 samples that are in test:
4 samples that are new:
087371
236487
113657
306898
```

Examples of performance on different types of networks with early stopping:

```
==> bigram-add.txt <==
step 15500 train loss: 2.3472518920898438 test loss: 2.348942995071411
2.3487308025360107 is best loss at step 15500, Early Stopping Applied. tag:
0x7fd1c8

==> bow-add.txt <==
step 83500 train loss: 1.5880573987960815 test loss: 1.6309101581573486
1.6254030466079712 is best loss at step 83500, Early Stopping Applied. tag:
0x28a712

==> gru-add.txt <==
step 35000 train loss: 1.1709833145141602 test loss: 1.2020294666290283
1.1984612941741943 is best loss at step 35000, Early Stopping Applied. tag:
0x9efce9

==> mlp-add.txt <==
step 27000 train loss: 1.1870310306549072 test loss: 1.2091768980026245
1.2088253498077393 is best loss at step 27000, Early Stopping Applied. tag:
0x341d29

==> rnn-add.txt <==
step 57000 train loss: 1.186924695968628 test loss: 1.2112380266189575
1.2109529972076416 is best loss at step 57000, Early Stopping Applied. tag:
0xf9158a

==> transformer-add.txt <==
step 14500 train loss: 1.1640605926513672 test loss: 1.1929705142974854
1.1849244832992554 is best loss at step 14500, Early Stopping Applied. tag:
0x6a33da
```

See Appendix 2 + 2A for a full example of optimization using tagging, also model sizing with the addition dataset.

TODO: How about adding bigger numbers using a larger dataset. How hard is for a transformer. I know it gets hard for a human to do this for bigger numbers, how about a transformer?

Sqrt

After that, I went to something a little bit more complicated, and that was to get square roots. And that involved a little bit more formatting. There's a colon and a space in this example.

There is a Python helper function to generate the square root data. The output from that code can be dumped into a text file, (ints-and-roots.txt) which it is in this archive. What that does is it takes numbers from 1 to 1000. The format includes a colon, and a space, and then to four decimal places has the square root of the integer number in front of the colon. And once again, you run that through makemore, and it does a pretty good job of learning it. In the new ones that it creates, where it has not seen them in the training data, most of the time, it surprisingly gets them correct or close. There's always a few in there that it's not close, it's out of bounds, but the most part, it can guess square roots pretty good. I expected worse from it.

If you run it long enough, it will just wind up memorizing the data set as well just like in the case for the addition example.

It's a pretty small dataset.

sqrt has a few more symbols than addition…

```
number of unique characters in the vocabulary: 13
vocabulary:
 .0123456789:
```

Examples of sqrt's that were generated. Some are far off base, others are

surprisingly close to the real value. Also notice how the colon gets thrown

in at the end in a few of them…

```
9 samples that are new:
978: 31.3169
970: 31.149:
682: 26.5543
447: 21.2038
682: 26.5543
96: 9.79:
968: 31.1161
4446: 2.0915
133: 11.5327
```

English Words Dataset, sorted_wordlist.txt = about 25x larger than names.txt at 6M characters

The other thing I tried was a much larger data set, about 25x larger than names.txt. I went and scoured three sources…

https://github.com/dolph/dictionary/blob/master/enable1.txt
http://wiki.puzzlers.org/pub/wordlists/unixdict.txt
https://github.com/dwyl/english-words/blob/master/words_alpha.txt

```
enable1.txt
unixdict.txt
words_alpha.txt
```

I did a concatenation using a sort with the -u option, which sorts all three that you can concatenate and sort them together. The -u option only produces a list that has a unique

ones. So, there is overlap in these three lists, so I wanted all of them together, but unique.

```
cat  enable1.txt unixdict.txt words_alpha.txt | sort -u > sorted_wordlist.txt
```

Training on that is interesting. It does produce interesting English-like words, the data set is larger, so it tends to take longer to train, tends to memorize as well but not as aggressively as when using names.txt. You will notice when running it there's more new output for a longer period of time.

It's kind of interesting to play with, as it comes up with some humorous words.

```
0xcb0e8e {'input_file': 'sorted_wordlist.txt', 'work_dir': 'wordlist', 'resume':
False, 'sample_only': True, 'num_workers': 4, 'max_steps': -1, 'device': 'cpu',
'seed': 3407, 'top_k': -1, 'type': 'transformer', 'n_layer': 4, 'n_head': 4,
'n_embd': 64, 'n_embd2': 64, 'batch_size': 32, 'learning_rate': 0.0005,
'weight_decay': 0.01, 'lt_thr': 10000000000.0, 'early_stop': 10000000000.0}
number of examples in the dataset: 548348
max word length: 31
number of unique characters in the vocabulary: 39
vocabulary:
&'.0123456789abcdefghijklmnopqrstuvwxyz
split up the dataset into 547348 training examples and 1000 test examples
dataset determined that: vocab_size=40, block_size=32
number of parameters: 0.20M
model #params: 207232
resuming from existing model in the workdir
--------------------------------------------------------------------------------
13 samples that are in train:
squallier
couchings
fireworm
cerebration
stinkier
message
spoon
hypothesize
kol
nous
discording
gobble
redesigning
0 samples that are in test:
37 samples that are new:
comprecise
arrogenry
ckikeke
revotial
sinharmic
baroody
hypsopacist
rakelings
subincreasing
buxblock
obsequiositousness
```

```
toreneting
involutionaries
butarises
pityoko
obliqua
warmet
dataras
vams
baracupalectomy
bized
auslandedly
nonsupercomplaiable
passar
cordit
gelluble
heliornitis
gibbonid
dendrosophy
supermarbital
intravagating
overamary
overdifficution
arto
valancy
sorptivity
interrusting
```

On Sizing Models

For what I have shown in this body of text makemore has been used with default settings…

```
'type': 'transformer', 'n_layer': 4, 'n_head': 4, 'n_embd': 64, 'n_embd2': 64,
'batch_size': 32, 'learning_rate': 0.0005, 'weight_decay': 0.01
```

...which works fine but, it is possible to size the models more in line with the datasets. In Appendix 2 and example is covered for the addition dataset which is very small at 70K file size/tokens. But, this example can be extended to other sizes easily enough.

See also nanoGPT (https://github.com/karpathy/nanoGPT)
https://github.com/karpathy/nanoGPT/blob/master/scaling_laws.ipynb
https://github.com/karpathy/nanoGPT/blob/master/transformer_sizing.ipynb

For sizing of the models. See if getting the model size closer to what can be calculated by formula. Also look at https://github.com/karpathy/llama2.c at this table for hints for starters...

model	dim	n_layers	n_heads	n_kv_heads	max context length	parameters	val loss	download
260K	64	5	8	4	512	260K	1.297	stories260K
OG	288	6	6	6	256	15M	1.072	stories15M.bin
42M	512	8	8	8	1024	42M	0.847	stories42M.bin
110M	768	12	12	12	1024	110M	0.760	stories110M.bin

… and the section that follows called **brief training guide**. Use this information to size the

models more correctly for the size of the datasets addition,sqrt,sorted_wordlist and 10M Quotes dataset.

See Appendix 2 + 2A for more information on model sizing.

Final 10M Quotes file

“The best was happening to some aim to talk the political.” -- makemore quotes

And finally, there's a file of one line quotes test where it's kind of throw the whole thing at it. A 10-meg file of quotes, there are sentences with punctuation, capitalization. It's got all letters, caps, no caps, punctuation, numerals. 81 distinct characters. This is kind of a just a push to see how far it can go. It's probably not going to do well with it, but it's a just run it and see what happens. I decided to run it under a distil-GPT-2 model as well and transfer learn on that just to see how it does with transfer learning on top of distil-GPT-2. TODO: Results.

It's running and running and running, and the loss keeps declining.
It's not memorizing any of it really at all due to the complexity of the data. It's just learning it to a degree. So, that's kind of a long duration run. Still TBD

Source for quotes:
https://www.researchgate.net/publication/304742521_CSV_dataset_of_76000_quotes_suitable_for_quotes_recommender_systems_or_other_analysis

It is a CSV file and just opened it up, removed the names column , and some other data column out of it just to get the plain quotes, to minimize confusion for the model. Ran that through, makemore, every step is sluggish compared to the other runs. Memory usage is obviously higher as well, something to watch for.
It eats more memory. If you're using it on a GPU or something that may have smaller RAM, you have a memory constraint to be mindful of.

Some samples….

```
step 200 | loss 1.7556 | step time 4900.78ms
--------------------------------------------------------------------------------
0 samples that are in train:
0 samples that are in test:
10 samples that are new:
As a nothth there Frecation that wo compertivention by world expriove struem sex
eses buttjuces wich of fine, of low willl aks anytearipton, he have and mak.
Prassily for man Goressss that can sunyed.. Aas. the hadound, shouldl of like is
 bagert rutmiction geat.
Whil the goe lave nothy to goals the fould in to beie I man'tefics, as wen mand
and cass. I down't whing musouse cosary.
We is people traing loving aged ours be imade to that scallle for trampictitmori
s opoicty.
The dan I man kidelling tragget And and art, Jandod the with ranesess grout surs
e of whel to where whicante.
As sempath wen us bellicures arsely equrilence make they.
Id pubateds be our the butting or moted have gelight do or latwered to bestaum a
nd not one fidulinet faity baind someration and Portie mydirghtoodys.
There sone affuriove domepers.
I went as I wante part forminevity we fuchth hear low.
```

I gliiketly it sitimes. Meay itwe no partsena hape mone taselly from over to a p
oshiction.

step 10000 train loss: 1.3898899555206299 test loss: 1.4004358053207397
test loss 1.4004358053207397 is the best so far, saving model to out/model.pt ta
g: 0x383047
----
0 samples that are in train:
0 samples that are in test:
10 samples that are new:
Hope people winning the latter of hold in the risk energy exponding a time is wo
rrited, and the way of them food. Pure never as going on hons or to leasties wit
hout, live reportions carress, and national him of weses phistory on within love
 is really learnity anageabily.
Drunk is a kid educatious.
Gimality is not now, vapkent and ignetting about. To wins you dreams, the is a y
our and when I'm talets against myself us a with.
I don't bobters anation when I was for a vailua lott and in the supparated progr
ess for.
We a soon alone couragy is solution, hope been in their friends old of love or d
efires as tempeptsive to purstable who contain political in. You excutive, consu
rity.
I would battle freedo all to home atte than country of the Higheraps decot bette
r, but the modern of broky a intertain, but caually denroism, has passionaters f
istantly over. And revelates, an terrol be bit. you just undeapable cchury take
around our passionable liberty can feelt of chingraiste sciesy sourcts.
I the start enough. It was loves I got that I would be folk story change to our
punneturally, but go conservoue been moving walk economical.
Music' understand, feach towarded oot movies on buy believe and our disacrimy. I
f is catined. It those gave it work out plare by to find I would success that is
 a lot.
I would believe where up in highest's writtenes.
I - my glore to yellainnd in be tooks richs, have worked, with talking Look one
-in angers at of a broad.

step 100000 train loss: 1.2169945240020752 test loss: 1.2245328426361084
test loss 1.2245328426361084 is the best so far, saving model to out/model.pt ta
g: 0x383047
----
0 samples that are in train:
0 samples that are in test:
10 samples that are new:
The gambles beauty of it is if you and you make me, I'm always from how you have
 a droid and willingp living, but blood or to ventus for years men into point. I
t's order in term 'Walthy and invited something by differentl.
Considered that I've always cycle worship neighboulking world with it these area
s serving evering of result, on Prostmartes at largely and capital the amount th
at we are pull as mucker, but - us quies to knows from your progression of mycat
ly: I kind of experience teaches of time is. Ough doesn't work it.
Intolery that single to a body, the struggle portraits as dinner so life.
Spressuits a little like each an idealism and greatly is person and not nowlic h
umorous behavior that I'd lucky, I'm always know there are quality for hygatisar
ously. You're compared. That disept even getting away, is cool intelligence with
 thems the day of the same two context modest if never poindle occased and long-
tweeats, pay fulled in underntright a gindeatwholizics,, I Carearcheastst.
I think it's success hard to everything and he means haven all that half taken.C

```
 Overnfularter is the entagery government that one student the granders and gett
ing blumb, to just estall natural intervirance.
I like the death of the gentleman.
The apports, and any delicately. To me he will gave us the hero porting.
Levetrates are demands clearly don't experience and be very declained in virtue.
. This idemistence is a very laugh success in the most growth of just disciples
in a motalf.
In dangle of the gay of a great change the concess of American language madificu
lty and hold taught to go to no one of their scrabble, the until it. So said, is
 a moman life's benefit.
You know the three government as something. It should look against true. Men and
 I had get into a young hostify changed people is certain. I mean film you would
 be in any stet, it imay they race, a set arts to the mother so writing.
```

I kept it running in the background of an old computer until about a month later the utility company turned off the power to upgrade the lines on the road….

```
step 828000 train loss: 1.1277523040771484 test loss: 1.1566824913024902
```

…

```
step 828200 | loss 1.1724 | step time 3191.39ms
----------------------------------------------------------------------------
0 samples that are in train:
0 samples that are in test:
10 samples that are new:
I love spending the whole decades, as a couchy.
He's what great political after someone we maguare if one matters.
It was the newspaed our romantic between duty. In this New Gesoton to Bart I was
gotten to make her sad effect with my mom was together, all the Americans that was
learning marriage was more he happiness.
It is eating moral, but one two years against the effort American proceeds looks
back to a master once.
I have early played to be drying a long short of itself, but we like a carner. It's
about deternity and women and another times in the ability part of the darkness.
Goodway is a houtcade moral system progressmands, it will spend a colvue-dinagner's
rolling. I think I know in the power to myself or the eyes on the certain drivers
than it's like to die.
The principle security, of well, or face ourselves, knowledge that they are no
population to encourage!
There is a job with an imitations of success can countered to constitution in
individual, disorderable into learning from Americans seldom and never has
competitions and the emotions of the decades in eduring produces for set and the
flaughter unementalement.
A lot of John Venundable son? The romantic consequences in they are, taking swords,
coortes that it just because they are very still desire artistic. That's important
as we as a rainly way to kind one to respect anyone - very ask men, great
felligence neithertiest in freedoms. The questions, it is at his great deal-
audience that.
The opparant of stereotype, for mere it. We are.
```

...it’s almost to the point where it has fair idea of spelling.

Note: If you pipe the output to a file or use nohup as was my case, you can find quotes via a pseudo inference. Like using grep “The best” found the quote at the top of this section. It creates so much output from training that just querying it with grep finds some funny and

interesting text.

grep “Cat “ nohup.out…

```
Cat from - my dad, even must judgling, or whether and intelligence.
The Cat of Lerr, I look is a rock reaches in Irandtool.
```

Grep on Love and then cherry pick some of the more interesting ones….

(See more examples in Appendix 3)

```
Love most never when minds and and priority.

Love means fellow.

Love is in way the power opposed of the great sexy.

Love no diety is by the starter that may be believe and look.

Love men may be call fear of God from - also attack world.

Love serious capable education is the such skin.

Love is one into God go to period environment at the family is those other
greatest.

Love to reform and unable full because, as blind presented for lead.

Love one's happiness, but made stressed it is to learn away that people thinking in
each of love by that.

Love is life is to become he hope, but in the heroism.

Love is the condition, every all men alone, we've great every third hope for it.

Love that stick would provide, more of money consider than the history of
respected, by his flowers him differents.
```

Model Size? And other questions that remain TBD

The quotes run above used the defaults...

```
0x383047 {'input_file': 'quotes_all.txt.csv', 'work_dir': 'out', 'resume': True,
 'sample_only': False, 'num_workers': 4, 'max_steps': -1, 'device': 'cpu', 'seed
': 3407, 'top_k': -1, 'type': 'transformer', 'n_layer': 4, 'n_head': 4, 'n_embd'
: 64, 'n_embd2': 64, 'batch_size': 32, 'learning_rate': 0.0005, 'weight_decay':
0.01, 'lt_thr': 10000000000.0, 'early_stop': 10000000000.0}
number of examples in the dataset: 75966
max word length: 505
number of unique characters in the vocabulary: 81
vocabulary:
 !"#$%&'()*+,-./0123456789:=?ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxy
z
split up the dataset into 74966 training examples and 1000 test examples
dataset determined that: vocab_size=82, block_size=506
number of parameters: 0.24M
```

```
model #params: 242944
```

Finally: Using even a 5x larger model which is more in line with the 10M dataset size definitely helps and results in lower loss in days.

Command: python3 makemore.py -i quotes_all.txt.csv -o quotes -l 0.001 -w 0.001 --n-head 5 --n-embd 125 --n-embd2 32 --n-layer 5 -b 32 --max-steps 100001 -e 10

```
0xcb2f85 {'input_file': 'quotes_all.txt.csv', 'work_dir': 'quotes', 'resume':
False, 'sample_only': Fa
lse, 'num_workers': 4, 'max_steps': 100001, 'device': 'cpu', 'seed': 3407, 'top_k':
-1, 'type': 'trans
former', 'n_layer': 5, 'n_head': 5, 'n_embd': 125, 'n_embd2': 32, 'batch_size': 32,
'learning_rate': 0
.001, 'weight_decay': 0.001, 'lt_thr': 10000000000.0, 'early_stop': 10}
number of examples in the dataset: 75966
max word length: 505
number of unique characters in the vocabulary: 81
vocabulary:
 !"#$%&'()*+,-./0123456789:=?ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
split up the dataset into 74966 training examples and 1000 test examples
dataset determined that: vocab_size=82, block_size=506
number of parameters: 1.02M
model #params: 1029625
step 0 | loss 4.5806 | step time 6334.96ms
```

....Current results...

```
step 52500 train loss: 1.0250550508499146 test loss: 1.0887385606765747
step 52510 | loss 1.0101 | step time 9953.35ms
step 52520 | loss 1.0745 | step time 9725.92ms
step 52530 | loss 1.0170 | step time 9717.88ms
step 52540 | loss 0.9489 | step time 9867.07ms
step 52550 | loss 1.0404 | step time 9408.16ms
step 52560 | loss 1.0207 | step time 9482.22ms
step 52570 | loss 1.0410 | step time 9732.82ms
step 52580 | loss 1.0141 | step time 9192.16ms
step 52590 | loss 1.0553 | step time 9614.88ms
step 52600 | loss 1.0285 | step time 10190.16ms
--------------------------------------------------------------------------------
0 samples that are in train:
0 samples that are in test:
10 samples that are new:
Thank with many sands has assumed the fields we find so little more unless attempts
to over-sighter has been said by Palm.
I get visual. I want to set your doctor for me. He wants when you are to like that,
where I don't, I am karing about all the 'person who knows that I'd be as a cool,
but I'll keep that sort of one enjoyable hault.
The conservatish deprised on politics is the best writered for bad, not a time but
in the war of an entire teaching us with the vanity.
Though I was writing that was always inspired 1978 who wanted to find a cambuck of
a joke. I like the past pools to go through, so I used for two months I had to
dress. It happens to bring me good. I did want to start younger to live there and I
would have been a high-great experience for God.
I am working on Knome, holes I'll be busy worrying about the 'E.R.' but I'm in a
sort of movie that I'm telling when it's happening with else because I felt what is
we can do it. If there's a non-typother publisher on it.
```

```
If you have an excessfus fellow of your children at some faith in your life's act,
that you do not any poetry.
Yesterday evil I can give my relationship with greatest death.
Let be lovers of victory and humiliating happiness, because they feel most unlored.
We are by the right to say that our time is also managed to survive ignorance.
Action we provision to work our hearts to positive our activities to our gifts and
occur.
On Christian superfluous offensive ideals, you come from , as youth personality
protectors, answer each other job in success, you have on some truth. And your
soul, do all your heart and light it is something that you might see it never asked
to a form of problem.
The fullness of our actor, not the education of the political road to are strong,
either wastandary for the qualities that we slict for which the walls of the
vastness becomes two things.
```

Notes/ TODO:

1. Probably could have a better train/test split. - Put in 10% for up to 100000 cases.
2. Model size could be better. Could pick something better than default settings. See Appendix 2.
3. Max word length 505, seems like block size is not capturing this but most quotes are much smaller so probably not an issue.
4. Probably a better candidate for training under nanoGPT, llama-2 train.py code from https://github.com/karpathy/llama2.c and/or transfer learn from distil-GPT2.
    4A. Did a tranfer learn on distil-GPT2, see Appendix 4
5. Good idea would be for nanoGPT- EX3: Find a dataset that is very large, so large that you can't see a gap between train and val loss. Pretrain the transformer on this data, then initialize with that model and finetune it on tiny shakespeare with a smaller number of steps and lower learning rate. Can you obtain a lower validation loss by the use of pretraining? (From: Let's build GPT: from scratch, in code, spelled out. https://www.youtube.com/watch?v=kCc8FmEb1nY)
6. Could mod inference code to be a 'calculator'. Take in 0-9 + - / * from input `and then feed that in as prompt and print one output from model. TDB` *- EX2: Train the GPT on your own dataset of choice! What other data could be fun to blabber on about? (A fun advanced suggestion if you like: train a GPT to do addition of two numbers, i.e. a+b=c. You may find it helpful to predict the digits of c in reverse order, as the typical addition algorithm (that you're hoping it learns) would proceed right to left too. You may want to modify the data loader to simply serve random problems and skip the generation of train.bin, val.bin. You may want to mask out the loss at the input positions of a+b that just specify the problem using y=-1 in the targets (see CrossEntropyLoss ignore_index). Does your Transformer learn to add? Once you have this, swole doge project: build a calculator clone in GPT, for all of +-*/. Not an easy problem. You may need Chain of Thought traces.) (From: Let's build GPT: from scratch, in code, spelled out.* https://www.youtube.com/watch?v=kCc8FmEb1nY)

See Appendix 2A for a grid search on hyper parameters for a "Tiny Math" model that was trained on math consiting of addition, subtraction, multiplication and division on numebr 0-9, excluding those that result in div/0.

7. GPT Eraser. I have tried to wipe the distil-gpt-2 model of it's memory using an entropy file consisting of random ASCII form 0-255 unsigned char. Can a model be partly wiped, lightly wiped, what happens? Open questions, TBD.

8. Go further with the Tiny Math examples. What about larger datasets greater than covering 0-9. Is there some point where the transformer model gets smarter and has emergent behavior. Just like how GPT-2 can just about talk and GPT-3 is coherant. But, small models trained on TinyStories seem coherent when ‘speaking’ like a 3-4 year old.

https://arxiv.org/abs/2305.07759
https://huggingface.co/datasets/roneneldan/TinyStories
https://github.com/karpathy/llama2.c

Conclusion

So, that is the modifications and other fun on makemore.py.

The zero to hero series,(https://github.com/karpathy/nn-zero-to-hero) which got me into looking deeper into makemore.
I started on the series, started looking at the code, looked at micrograd.
And for some reason, just start to play with makemore, and this is the result of it.

Hope you enjoy it.
Have fun.
Play with the code.
Learn something.

NOTE: This is a verbose README as I spoke this and used Whisper to convert to text, then edited….quite a bit and added more later on.
Then fed this in ChatGPT with the foloowing prompt to get a reasonable README.md out. ChatGPT was able to produce markdown code directly. This seems to be an improvement as I tried this once before months ago and it was not successful despite prodding it several ways.

"Can you proofread and edit the following text, it is going to be used as a README.md file for a Github fork of the makemore project.... This code is a fork of Andrej Karpathy's makemore, it's there to try to, tune hyperparameters for makemore and experiment with smaller and larger datasets than names.txt"

Appendix 1

Helper code

```
gen-sqrts-for-train.py - Generate sqrts of integers from 1-1000, to 4 places
seperated by colon and space.
addition-example.py - Generate addition examples, in a format where it is just
integers zero padded, result is right hand digits. Ex: 010102, 112233,122436.


batch-search.sh - Find fastest performing batch size.
grid-search-names-layer-emb-layers.sh - Do a grid search for optimization.
grid-search-names.sh - Do a grid search for optimization.
gs-math-dataset.sh – Example grid search on math dataset.
```

X grid-search-orig.sh
grid-search.sh - Do a grid search for optimization.
initialization-lottery-test.sh – Runs the initialization lottery test as described in the paper. Chooses random seeds and runs makemore looking to see how initilization affects loss after 500 steps.
math_dataset_gen.py – Generates examples of math, addition, subtraction, multiplication and division.
optimization-lottery-test.sh - Try different seed values to see how it affects performance.
plot-makemore-learning-curves.py – Takes the file makemore_log.csv and plots train and test losses against steps. This makemore_log.csv is appended by default. For multiple runs, either delete it. Or use the hex tag for the run to grep on and parse out the run from it to another filename and plot with that. Probably could MOD plot-makemore-learning-curves.py to take optional different filename TBD.

sample-only.sh - Sample names
sample-wordlist.sh - Sample from the large wordlist.
train-on-names-optimized.sh - Train with hyperparameters found by optimization, via grid search.
train-on-names.sh - Train on names helper
train-on-sorted-wordlist.sh - Train on the large wordlist.

Datasets

enable1.txt – English words.
unixdict.txt - English words.
math_dataset.txt – Math on 0-9, examples of + - * / on numbers from 0-9, random, like...
2 + 3 = 5.00
6 + 1 = 7.00
2 + 3 = 5.00
8 - 1 = 7.00
9 + 6 = 15.00
3 + 7 = 10.00
6 + 2 = 8.00
4 + 4 = 8.00
3 * 3 = 9.00
8 * 9 = 72.00

sort_u_math_dataset_100.txt – Same as above but 0-99 and sorted to unique samples.
words_alpha.txt - English words.
sorted_wordlist.txt – Made using … cat enable1.txt unixdict.txt words_alpha.txt | sort -u > sorted_wordlist.txt
addition.txt - 10000 random addtion examples. start 010102 end 9901100
ints-and-roots.txt 20000 sqrt examples. 1-1000, without zero leading padding. Format colon and space between. Answers to 4 decimal places.
sort_u_math_dataset_100.txt – Math on 0-99 but used sort -u to make a set that is not random and ONLY has one example for each. Creating a random dataset that captures every example results in a rather large dataset and this gets all the examples just once.

Optimization Examples
addition-test.txt
names-test.txt

Appendix 2

Tags and Optimization Example and Model Sizing

Tags and Optimization

Using the gs-addition.txt that is the output created by piping the output from the gs-addition.sh to that file.

```
./gs-addition.sh > gs-addition.txt
...after it is sone running...
grep best gs-addition.txt | sort -r | tail
test loss 1.3630998134613037 is the best so far, saving model to gridtest/model.pt
tag: 0x16ca1f
test loss 1.3549249172210693 is the best so far, saving model to gridtest/model.pt
tag: 0x8570dd
test loss 1.3323713541030884 is the best so far, saving model to gridtest/model.pt
tag: 0xdd754
test loss 1.3206613063812256 is the best so far, saving model to gridtest/model.pt
tag: 0x480414
test loss 1.3141957521438599 is the best so far, saving model to gridtest/model.pt
tag: 0x7d2128
test loss 1.3077924251556396 is the best so far, saving model to gridtest/model.pt
tag: 0x159129
test loss 1.291678786277771 is the best so far, saving model to gridtest/model.pt
tag: 0xdeb431
test loss 1.2915117740631104 is the best so far, saving model to gridtest/model.pt
tag: 0x84ed60
test loss 1.2740612030029297 is the best so far, saving model to gridtest/model.pt
tag: 0x6ae8bd
test loss 1.2582876682281494 is the best so far, saving model to gridtest/model.pt
tag: 0xabcb26
```

The best performer, 1.2582876682281494 is at the bottom of the list. Before looking at it closely. Here is top of the sort in the other direction.

```
grep best gs-addition.txt | sort | tail -1
test loss 2.401581048965454 is the best so far, saving model to gridtest/model.pt
tag: 0xdbd5
```

The loss of about 2.4 tells us that not a lot of learning has happened. Why is this? Well, it is easy to calculate the natural entropy in picking 1 token from a group of 10 possibilities for the numbers 0-9 used in the addition code. Basically we just have to calculate the negative log likelihood using -ln(1/10). The result is 2.302585093. So at the level it is just a random guess picking one token out of 10, kind of like monkeys typing! And, as can be seen some examples even start out around this number

at step 0…

```
grep "step 0" gs-addition.txt | sort -r | tail
step 0 | loss 2.4266 | step time 32.71ms
step 0 | loss 2.4265 | step time 70.11ms
step 0 | loss 2.4265 | step time 66.92ms
step 0 | loss 2.4265 | step time 61.16ms
```

```
step 0 | loss 2.4265 | step time 57.10ms
step 0 | loss 2.4265 | step time 43.51ms
step 0 | loss 2.4265 | step time 40.25ms
step 0 | loss 2.4265 | step time 36.68ms
step 0 | loss 2.4265 | step time 147.40ms
step 0 | loss 2.4265 | step time 107.35ms
```

Back to the best case…

test loss 1.2582876682281494 is the best so far, saving model to gridtest/model.pt tag: 0xabcb26

...using grep with the tag value for search will reveal the hyperparameters that were used in this run.

```
0xabcb26 {'input_file': 'addition.txt', 'work_dir': 'gridtest', 'resume': False,
'sample_only': False, 'num_workers': 4, 'max_steps': 501, 'device': 'cpu', 'seed':
3407, 'top_k': -1, 'type': 'transformer', 'n_layer': 1, 'n_head': 8, 'n_embd': 32,
'n_embd2': 32, 'batch_size': 32, 'learning_rate': 0.01, 'weight_decay': 0.001,
'lt_thr': 10000000000.0, 'early_stop': 10000000000.0}
```
test loss 1.2582876682281494 is the best so far, saving model to gridtest/model.pt tag: 0xabcb26

```
# Define the range of values for each hyperparameter
param_1=(1e-2 1e-3 1e-4) # lr
param_2=(0.1 0.01 0.001) # weight decay
param_3=(1 2 4 8) # N-heads for GPT
param_4=(8 16 32) # Embed 1
param_5=(32) # Embed 2
param_6=(1 2 4 8) # Layers
```

As some of the values are at the extremes, it might pay to rerun with ranges centered around those values. Maybe weight decay could be lower? Heads higher? Embed higher? I have noticed that the transformer doesn’t seem to care about embed 2, so I just fixed it to 32.

Tried it with a range as follows but the same hyperparameters won out again.

```
# PASS 2
param_1=(0.5e-1 1e-2) # lr
param_2=(0.001 0.0001) # weight decay
param_3=(4 8 16) # N-heads for GPT
param_4=(16 32 64) # Embed 1
param_5=(32) # Embed 2
param_6=(1 2 4) # Layers
```

Using CSV file

The recently added ability to create the makemore_log.csv allows a sort of the test loss to

reveal the best performer in a grid search as show in the following example where a grid search is performed on a run on the small 0-9 math set (math_dataset.txt).

sort -nk4 -t, -r makemore_log.csv | tail

```
0x324fef,500,0.47,0.501
0x119fbc,500,0.469,0.5
0x5ba887,500,0.469,0.499
0x3254ff,500,0.467,0.499
0xa5510a,500,0.468,0.498
0xfca52,500,0.469,0.497
0xbf86d,500,0.455,0.496
0xa5362b,500,0.455,0.496
0x9d98df,500,0.451,0.495
0x78a5b6,500,0.451,0.495
```

Taking the hex tag of the last line or trying a few as they have the same or nearly the same losses can check to see how other models perform…

```
grep 0x78a5b6 gs-gridtest.txt

0x78a5b6 {'input_file': 'math_dataset.txt', 'work_dir': 'gridtest', 'resume':
False, 'sample_only': False, 'num_workers': 4, 'max_steps': 501, 'device': 'cpu',
'seed': 3407, 'top_k': -1, 'type': 'transformer', 'n_layer': 4, 'n_head': 8,
'n_embd': 128, 'n_embd2': 32, 'batch_size': 32, 'learning_rate': 0.001,
'weight_decay': 0.001, 'lt_thr': 10000000000.0, 'early_stop': 10000000000.0}
test loss 0.49506667256355286 is the best so far, saving model to gridtest/model.pt
tag: 0x78a5b6

grep 0xa5362b gs-gridtest.txt

0xa5362b {'input_file': 'math_dataset.txt', 'work_dir': 'gridtest', 'resume':
False, 'sample_only': False, 'num_workers': 4, 'max_steps': 501, 'device': 'cpu',
'seed': 3407, 'top_k': -1, 'type': 'transformer', 'n_layer': 4, 'n_head': 8,
'n_embd': 128, 'n_embd2': 32, 'batch_size': 32, 'learning_rate': 0.001,
'weight_decay': 0.01, 'lt_thr': 10000000000.0, 'early_stop': 10000000000.0}
test loss 0.49610593914985657 is the best so far, saving model to gridtest/model.pt
tag: 0xa5362b
```

… same paramters,except for weight decay, highlighted in red.

Model Sizing

Are these parameters as in the size of the model reasonable? Using add_scaling_laws.py (mod of file from nanoGPT archive See nanoGPT (https://github.com/karpathy/nanoGPT) https://github.com/karpathy/nanoGPT/blob/master/scaling_laws.ipynb

https://github.com/karpathy/nanoGPT/blob/master/transformer_sizing.ipynb)

the following is calculated...

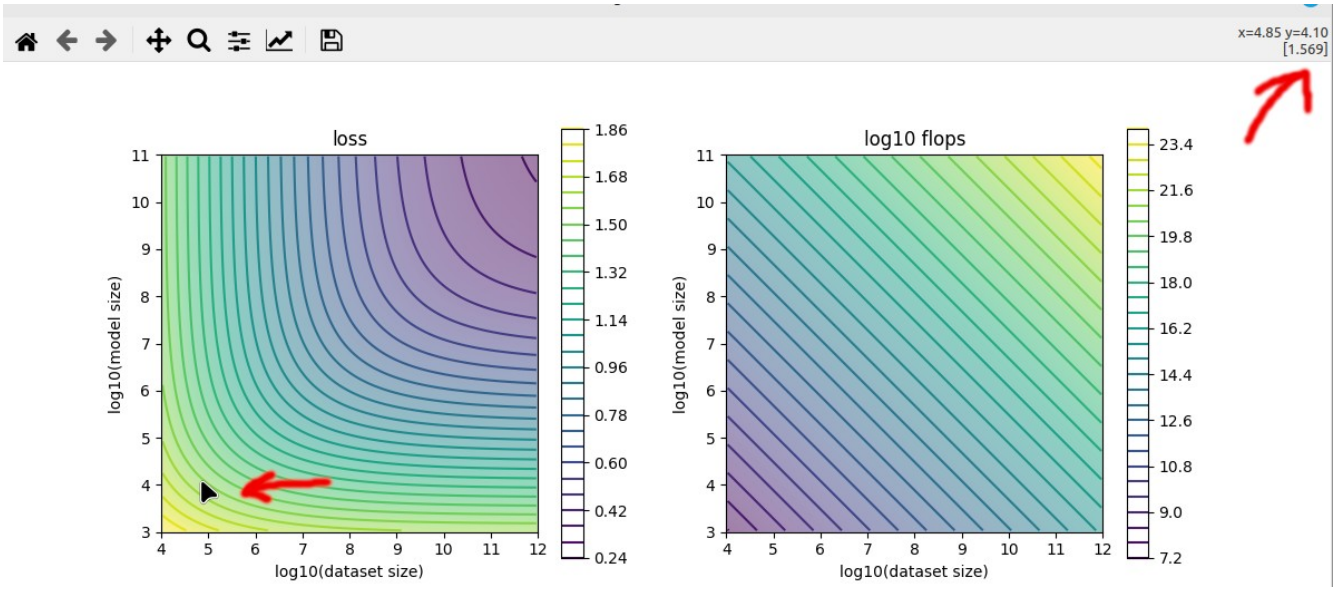
```
Model Parameters:  13120
y = 1.0409573169995885x + 0.935388715239086
predicted parameters for 1.312000e+04 tokens: 1.667178e+05
```

The addition.txt file is 70513 characters which is less than 1.667178e+05, but no by something like an order of magnitude.

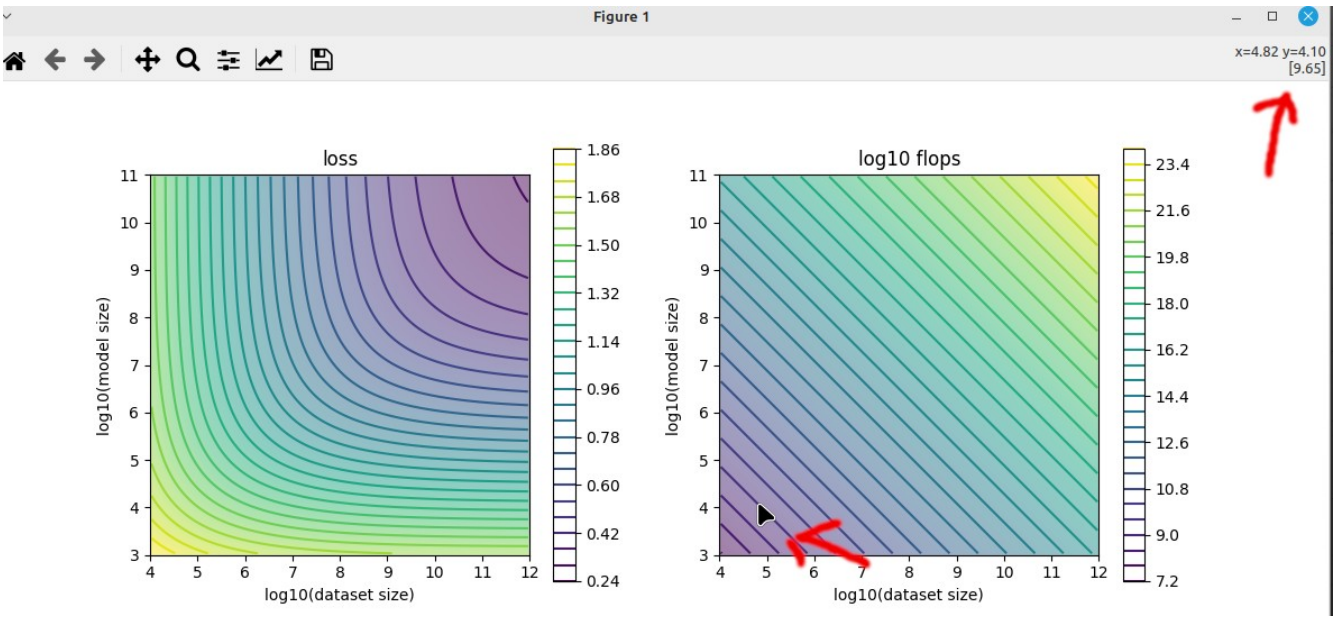
From scaling_laws_3_gpt2_small.py (also a modification of the fie from nanoGPT archive)

Interpolating from the graph is produces using the following for dataset size of tokens and the approx number of parameters...

```
>>> import numpy as np
>>> np.log10(70000)
4.845098040014257
>>> np.log10(13000)
4.113943352306837
```



Looking at the arrows which I positioned on the point on the graph that lines up with the parameters and tokens/dataset, we see a loss of 1.569. This I imagine is a good reasonable loss.



Positioning on the right hand chart with same location, it looks like compute would be 10^9.65 FLOPS

To confirm….

```
>>> 10**9.65
4466835921.509635
>>> 4.4e9
4400000000.0
```

4.4e9 FLOPS. On the unit I tried this on the CPU on an old machine (3rd gen i3) that seems to be good for about 25GFLOPS, so this should take….

```
>>> 4.4e9/25e9
0.176 seconds.
```

...It clearly does not do this, probably other loads on the system and inefficiencies, but opening up gs-addition.txt and finding the tag, shows the actual ground truth.

```
0xabcb26 {'input_file': 'addition.txt', 'work_dir': 'gridtest', 'resume': False>
number of examples in the dataset: 10000
max word length: 7
number of unique characters in the vocabulary: 10
vocabulary:
0123456789
split up the dataset into 9000 training examples and 1000 test examples
dataset determined that: vocab_size=11, block_size=8
number of parameters: 0.01M
model #params: 13728
step 0 | loss 2.5428 | step time 43.59ms
step 10 | loss 1.9168 | step time 14.34ms
step 20 | loss 1.8884 | step time 14.27ms
step 30 | loss 1.8209 | step time 14.22ms
step 40 | loss 1.8209 | step time 11.49ms
step 50 | loss 1.8221 | step time 17.22ms
step 60 | loss 1.8056 | step time 22.20ms
```

```
step 70 | loss 1.7490 | step time 15.70ms
step 80 | loss 1.8229 | step time 15.66ms
step 90 | loss 1.7565 | step time 15.90ms
step 100 | loss 1.7047 | step time 11.22ms
step 110 | loss 1.7079 | step time 12.37ms
step 120 | loss 1.7129 | step time 14.37ms
step 130 | loss 1.5471 | step time 10.88ms
step 140 | loss 1.6281 | step time 14.20ms
step 150 | loss 1.5766 | step time 10.91ms
step 160 | loss 1.6732 | step time 13.87ms
step 170 | loss 1.5051 | step time 11.04ms
step 180 | loss 1.5072 | step time 10.94ms
step 190 | loss 1.6157 | step time 14.36ms
step 200 | loss 1.6961 | step time 23.98ms
```

After step 130 to 180, it is clear that it is trying to get down in the range shown on the plot. Let's say it is good enough-ish at 200 steps. The quickest was to calculate the total ms, is just to copy/paste this into ChatGPT and ask it…

To calculate the total number of milliseconds for this run, you can sum up the step times. Here's the calculation:

1. Add up all the step times: 43.59ms + 14.34ms + 14.27ms + 14.22ms + 11.49ms + 17.22ms + 22.20ms + 15.70ms + 15.66ms + 15.90ms + 11.22ms + 12.37ms + 14.37ms + 10.88ms + 14.20ms + 10.91ms + 13.87ms + 11.04ms + 10.94ms + 14.36ms + 23.98ms
2. Calculate the total: Total = 357.84ms

So, the total number of milliseconds for this run is approximately 357.84 milliseconds.

It's not too far off from what the plot shows.

But at 200 steps it is still kind of junky for outputs, it gets 2 right and the new ones are bad….

```
2 samples that are in train:
590261
730477
0 samples that are in test:
8 samples that are new:
820182
930296
9601101
8710100
283567
491660
183146
711798
```

Next do a run to 10001 steps and pipe it to test.txt then use grep…

```
grep "test loss" test.txt | grep step
step 500 train loss: 1.2470526695251465 test loss: 1.2582876682281494
step 1000 train loss: 1.1927334070205688 test loss: 1.217165231704712
step 1500 train loss: 1.185994267463684 test loss: 1.199806809425354
```

```
step 2000 train loss: 1.1971455812454224 test loss: 1.208845615386963
step 2500 train loss: 1.1844254732131958 test loss: 1.1994160413742065
step 3000 train loss: 1.1833233833312988 test loss: 1.200137734413147
step 3500 train loss: 1.1846715211868286 test loss: 1.194693684577942
step 4000 train loss: 1.2359402179718018 test loss: 1.249329686164856
step 4500 train loss: 1.175779938697815 test loss: 1.190536618232727
step 5000 train loss: 1.192779541015625 test loss: 1.1909372806549072
step 5500 train loss: 1.1783798933029175 test loss: 1.1935014724731445
step 6000 train loss: 1.1963151693344116 test loss: 1.209836721420288
step 6500 train loss: 1.1742756366729736 test loss: 1.1903144121170044
step 7000 train loss: 1.1741561889648438 test loss: 1.1890884637832642
step 7500 train loss: 1.1831822395324707 test loss: 1.1969531774520874
step 8000 train loss: 1.1814955472946167 test loss: 1.1947546005249023
step 8500 train loss: 1.1816658973693848 test loss: 1.1969847679138184
step 9000 train loss: 1.180533528327942 test loss: 1.1913996934890747
step 9500 train loss: 1.1999298334121704 test loss: 1.214128017425537
step 10000 train loss: 1.1870163679122925 test loss: 1.1987128257751465
```

Around the point where the test loss and train loss become equal it is probably fair to say that the model is performing well. If the train loss is starting to be a lot lower than test loss, it is starting to overfit and it has pretty much memorized the dataset.

At step 4800 it seems to be doing better than at 200 and seems trained.

```
8 samples that are in train:
236184
920395
9604100
540357
411556
028486
761490
9901100
0 samples that are in test:
2 samples that are new:
191029
511970
```

Pushing it to 100000 steps you can see the train loss is clearly less than test loss. It still makes mistakes but this is to be expected…

```
step 100000 train loss: 1.1580466032028198 test loss: 1.1950645446777344
--------------------------------------------------------------------------------
8 samples that are in train:
710677
376299
540862
670774
683199
335285
781290
930699
0 samples that are in test:
2 samples that are new:
652186
256388
```

Early Stopping

The early stopping option tells the code to stop when the loss no longer drops for a number of cycles in the run. The cycles are 500, which is when the model is potentially saved, when the test loss improves. Using -e 1, means it will stop the code after 1 failure to show improvement in loss. This is type of threshold can prevent overfitting.

Using -e 1...

```
step 5500 train loss: 1.1783798933029175 test loss:
1.1935014724731445
```

```
1.190536618232727 is best loss at step 5500, Early Stopping Applied.
tag: 0x83866a
```

Seems like a decent place to stop the code as before we noted that around step 5000 both train and test losses were starting to become equal.

Pushing it a bit more using -e 10 for example…

```
step 24500 train loss: 1.1702595949172974 test loss:
1.1897960901260376
```

```
1.185193419456482 is best loss at step 24500, Early Stopping Applied.
tag: 0x969577
```

Note the tag, early stopping is tagged as well as regular saving of the best parameters allowing it to be seachable as well if a grid search is performed. The tag is on the end of the output, making it easy to use sort.

Early stopping does a reasonable job at finding a point to cut off training. This an easy way to keep the model from overfitting and it might generalize better. Dropout is also a way of helping this out but is not an option for makemore as it is meant to be used on small models where dropout just might not be worth bothering with but, it is something to be kept in mind for training larger models. NanoGPT for example has dropout available and this can be used to help models generalize better.

Appendix 2A

Optimizing while constraining model size

Taking a hint from TinyStories (https://arxiv.org/abs/2305.07759 https://huggingface.co/datasets/roneneldan/TinyStories https://github.com/karpathy/llama2.c

) , how about seeing how small of a model can do math, Tiny Math ??? Adding, subtracting, multiplying and dividing numbers from 0-9 in the following form…

```
2 + 3 = 5.00
6 + 1 = 7.00
2 + 3 = 5.00
```

```
8 - 1 = 7.00
9 + 6 = 15.00
3 + 7 = 10.00
6 + 2 = 8.00
4 + 4 = 8.00
3 * 3 = 9.00
8 * 9 = 72.00
```

…and constrain the model to just 2000 parameters for this file of…

```
cat math_dataset.txt | wc
  1000   5000  13443
```

… 1000 random math examples, 13443 bytes...

```
 cat math_dataset.txt | sort -u | wc
   345   1725   4628
```

… of which 345 are unique to add,sub,mul,div for 0-9, excluding div/0 cases. A model of 2000 parameters should be able to cope with this amount of data.

Searching using the following hyperparameters in (gs-math-dataset.sh)

```
param_1=(0.01 0.005 0.001 0.0005 0.0001) # lr
param_2=(0.01) # weight decay
param_3=(1 2 3 4 6 ) # N-heads for GPT
param_4=(4 8 12 16 24 32) # Embed 1
param_5=(32) # Embed 2
param_6=(1 2 3 4 6) # Layers
```

...more code…

```
python3 makemore.py -i math_dataset.txt -o gridtest -l $p1 -w $p2 --n-head $p3 --n-
embd $p4 --n-embd2 $p5 --n-layer $p6 --max-steps 501 -b 32 -m 2000
```

Note the 2000 for the model parameter constraint.

RUN the grid search and when done…

```
sort -nk4 -t, -r  makemore_log.csv | wc
90 -l
```

… there are 90 runs, so that seems like a reasonable sample,
looking at the best 10...

```
sort -nk4 -t, -r  makemore_log.csv | tail

0x1305c0,500,0.716,0.727
0x79b07d,500,0.716,0.726
0x2f86d1,500,0.704,0.726
0x9a1ac1,500,0.711,0.725
0xd6b61,500,0.71,0.72
0x6b4be9,500,0.691,0.704
0x949b9,500,0.679,0.69
0xe7f8d5,500,0.655,0.669
0xe5ead6,500,0.646,0.662
0x22e9ee,500,0.64,0.659
```

...use grep on the tag to find the best performer…

```
grep 0x22e9ee nohup.out

0x22e9ee {'input_file': 'math_dataset.txt', 'work_dir': 'gridtest', 'resume':
False, 'sample_only': False, 'num_workers': 4, 'max_steps': 501, 'device': 'cpu',
'seed': 3407, 'top_k': -1, 'type': 'transformer', 'n_layer': 1, 'n_head': 4,
'n_embd': 8, 'n_embd2': 32, 'batch_size': 32, 'learning_rate': 0.01,
'weight_decay': 0.01, 'lt_thr': 10000000000.0, 'early_stop': 10000000000.0,
'max_params': 2000}
```

...From nohup.out it can be seen that this is a…

```
model num. Params: 1288
```

...very small model with 1288 parameters, only 1 layer, 4 heads, 8 n embd, lr of 0.01

How does it work, using option -e 1, it does not train long enough pushing it to -e 10…

```
step 21800 | loss 0.4871 | step time 10.01ms
-------------------------------------------------------------------------------
8 samples that are in train:
5 + 5 = 10.00
2 - 2 = 0.00
9 + 8 = 17.00
0 - 2 = -2.00
0 + 9 = 9.00
1 * 5 = 5.00
3 / 2 = 1.50
1 + 1 = 2.00
0 samples that are in test:
2 samples that are new:
9 - 0 = 4.00
9 / 4 = 2.45    ←--- Nice try, close to the real answer of 2.25

…
step 22000 | loss 0.5192 | step time 8.37ms
step 22000 train loss: 0.4723145663738251 test loss: 0.5097182989120483
0.49580031633377075 is best loss at step 22000, Early Stopping Applied. tag:
0x6a7e74
```

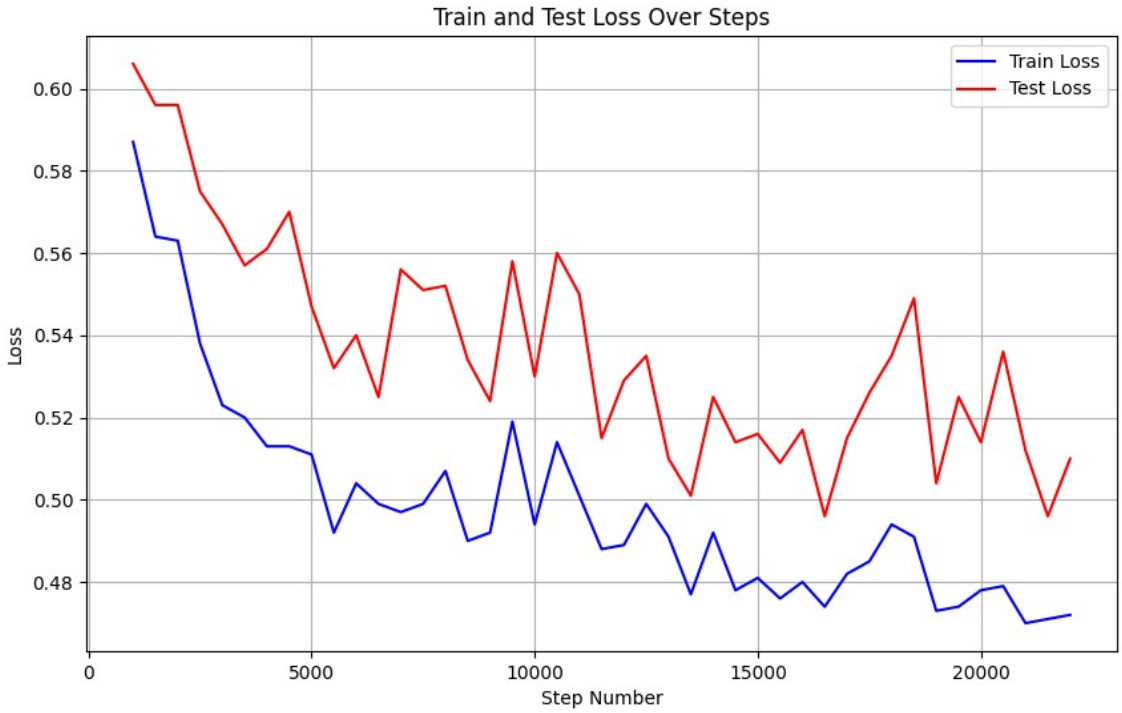
… it runs 22000 steps and the loss is under 0.5 down from an initial 2.8890. 17+1 characters, -ln(1/18) = `2.8903717579`
e^0.5 = 1.6487212707 so the odds are much higher now than the initial 1/18 chance of getting the next character right.
Looking above the model gets 8 right and 2 wrong. Anything new is not in the dataset and therefore a wrong answer. Scrolling up and looking at the run a bit, sometimes it almost gets them all right and sometimes very wrong…

```
3 samples that are in train:
7 * 3 = 21.00
9 + 5 = 14.00
6 + 9 = 15.00
1 samples that are in test:
5 - 2 = 3.00  ←-- Correct but not in dataset.
6 samples that are new:
```

```
2 * 3 = 8.00
3 - 5 = 1.00      <--- Almost right!
8 + 9 = 9.00
4 * 5 = 20.00   <--- Was not in original dataset and is correct!
4 / 9 = 0.59    <-- Nice try, really is 0.44444
8 + 5 = 14.00   <-- Close again!
```

It's like it's have a good day and then a bad day with math. But, the results are not much worse that what one would expect from an elementary school student just learning this kind of math. It also does not seem to learn one type better than another judging from the fact that addition,subtraction,multiplication and division all appear as wrong about equally often.
4 * 5 = 20.00 and 5 - 2 = 3.00 are correct but are not in the dataset, something to be aware of.

Finally using plot-makemore-learning-curves.py allows plotting of the data from the file makemore_log.csv



What about increasing the math dataset by 10x

This 10x change is made by increasing the range to 0-28. This will increase the size of the unique math problems in the dataset by 10. For this run, the unique problems are sorted to a file that contains only the unique and not a random collection of math problems from 0-28. This helps keep the file smaller and makes it easy to see if the answers are correct as all of

the combinations of 0-28 +,-,*,/ are represented in the dataset.

This time there are three candidates out of 125 model tests that have the same loss.

```
0xffcc31,500,0.825,0.823
0xdba8d,500,0.827,0.823
0x4c0273,500,0.83,0.823
```

```
 grep 0x4c0273 nohup.out
0x4c0273 {'input_file': 'math_dataset_28_sorted.txt', 'work_dir': 'gridtest',
'resume': False, 'sample_only': False, 'num_workers': 4, 'max_steps': 501,
'device': 'cpu', 'seed': 3407, 'top_k': -1, 'type': 'transformer', 'n_layer': 2,
'n_head': 6, 'n_embd': 12, 'n_embd2': 32, 'batch_size': 32, 'learning_rate': 0.005,
'weight_decay': 0.01, 'lt_thr': 10000000000.0, 'early_stop': 10000000000.0,
'max_params': 6000}
 grep 0xdba8d nohup.out
0xdba8d {'input_file': 'math_dataset_28_sorted.txt', 'work_dir': 'gridtest',
'resume': False, 'sample_only': False, 'num_workers': 4, 'max_steps': 501,
'device': 'cpu', 'seed': 3407, 'top_k': -1, 'type': 'transformer', 'n_layer': 2,
'n_head': 6, 'n_embd': 12, 'n_embd2': 32, 'batch_size': 32, 'learning_rate': 0.01,
'weight_decay': 0.01, 'lt_thr': 10000000000.0, 'early_stop': 10000000000.0,
'max_params': 6000}
grep 0xffcc31 nohup.out
0xffcc31 {'input_file': 'math_dataset_28_sorted.txt', 'work_dir': 'gridtest',
'resume': False, 'sample_only': False, 'num_workers': 4, 'max_steps': 501,
'device': 'cpu', 'seed': 3407, 'top_k': -1, 'type': 'transformer', 'n_layer': 2,
'n_head': 3, 'n_embd': 12, 'n_embd2': 32, 'batch_size': 32, 'learning_rate': 0.01,
'weight_decay': 0.01, 'lt_thr': 10000000000.0, 'early_stop': 10000000000.0,
'max_params': 6000}
```

2 layers, n_emdb 12, n_heads 3 or 6, lr 0.01 or 0.005 for 6 n_heads.

Running a test on 3 heads, LR 0.01,2 layers and 12 n_embd and early stopping at 10…

```
model num. Params: 4428

…

step 27800 | loss 0.6733 | step time 13.50ms
-------------------------------------------------------------------------------
0 samples that are in train:
0 samples that are in test:
10 samples that are new:
9 * 1 = 8.00
20 / 12 = 1.70
25 - 4 = 20.00
8 - 11 = -19.00
2 / 9 = 0.21
13 / 20 = 0.60
4 / 4 = 0.00
8 + 1 = 1.00
14 + 2 = 18.00
23 + 15 = 46.00
…
step 28000 train loss: 0.680976152420044 test loss: 0.7038818597793579
```

```
0.6959528923034668 is best loss at step 28000, Early Stopping Applied. tag:
0xcf187c
```

Poor performance this time for a model at 28000 steps with a model that has a parameter size that is about 1/10 the size of the dataset. Perhaps the model is just too simple to succeed at learning math across 0-28.

Trying 6 heads, which shows same model size…

```
model num. Params: 4428
```

…

```
step 14800 | loss 0.6496 | step time 18.42ms
-----------------------------------------------------------------------------
3 samples that are in train:
18 + 19 = 37.00
0 / 13 = 0.00
12 + 14 = 26.00
0 samples that are in test:
7 samples that are new:
9 * 23 = 31.00
7 * 3 = 30.00
10 + 14 = 25.00
28 / 25 = 1.26
1 - 13 = -13.00
13 / 9 = 1.68
14 / 10 = 1.27
```

… still would fail a math test!

Bumping up the model size and letting it run longer makes a difference…

3 layers

8 heads

16 n_embd

= Params: 10720

Letting it run using a big early stop of 100 and a loop limit of 100000 it hits the loop limit first.

```
0xda395d {'input_file': 'math_dataset_28_sorted.txt', 'work_dir': 'math-28',
'resume': False, 'sample_only': False, 'num_workers': 4, 'max_steps': 100001,
'device': 'cpu', 'seed': 3407, 'top_k': -1, 'type': 'transformer', 'n_layer': 3,
'n_head': 8, 'n_embd': 16, 'n_embd2': 32, 'batch_size': 32, 'learning_rate': 0.01,
'weight_decay': 0.01, 'lt_thr': 10000000000.0, 'early_stop': 100, 'max_params':
10000000000.0}
number of examples in the dataset: 3248
max word length: 16
number of unique characters in the vocabulary: 17
vocabulary:
 *+-./0123456789=
```

```
split up the dataset into 2924 training examples and 324 test examples
dataset determined that: vocab_size=18, block_size=17
number of parameters: 0.01M
model num. Params: 10720, model size (model.py) 58309
```
As can be seen above the model size is 58309 and exceeds the the dataset (math_dataset_28_sorted.txt) size of 49384 by a lot. So in reality code that runs a simple lookup would be the best solution! There is no compression advantage gained by using an ML model over a look up table in this case.

```
...


step 100000 | loss 0.5739 | step time 21.11ms
step 100000 train loss: 0.5957510471343994 test loss: 0.631156861782074
--------------------------------------------------------------------------------
7 samples that are in train:
19 + 3 = 22.00
28 + 5 = 33.00
1 * 28 = 28.00
28 + 13 = 41.00
16 + 18 = 34.00
17 + 25 = 42.00
0 * 2 = 0.00
0 samples that are in test:
3 samples that are new:
13 / 5 = 2.75  ←-- close, actual is 2.6
18 * 9 = 152.00   -- close, actual is 162, seems like it gets middle digit off by
one.
11 * 17 = 177.00  -- close, actual is 187, seems like it gets middle digit off by
one.
```

… gets 3 wrong, so it would get a passing grade on a ‘test’ in school at least.

Finally let’s do a gird search again. But now the model sizes max of 16000 parameters is really going to exceed the the dataset (math_dataset_28_sorted.txt) size of 49384 by a lot so it’s not practical but we will do it for a toy exercise.

```
Doing the grid search the best model picked is just shy of 16000 parameters…
0x758803 {'input_file': 'math_dataset_28_sorted.txt', 'work_dir': 'math-28', 're
sume': False, 'sample_only': False, 'num_workers': 4, 'max_steps': 100001, 'devi
ce': 'cpu', 'seed': 3407, 'top_k': -1, 'type': 'transformer', 'n_layer': 2, 'n_h
ead': 12, 'n_embd': 24, 'n_embd2': 32, 'batch_size': 32, 'learning_rate': 0.01,
'weight_decay': 0.005, 'lt_thr': 10000000000.0, 'early_stop': 100, 'max_params':
 10000000000.0}
```

2 layers, 12 heads, 24 n_embd, lr of 0.005.

model num. Params: 15768, file size of model.pt 74310

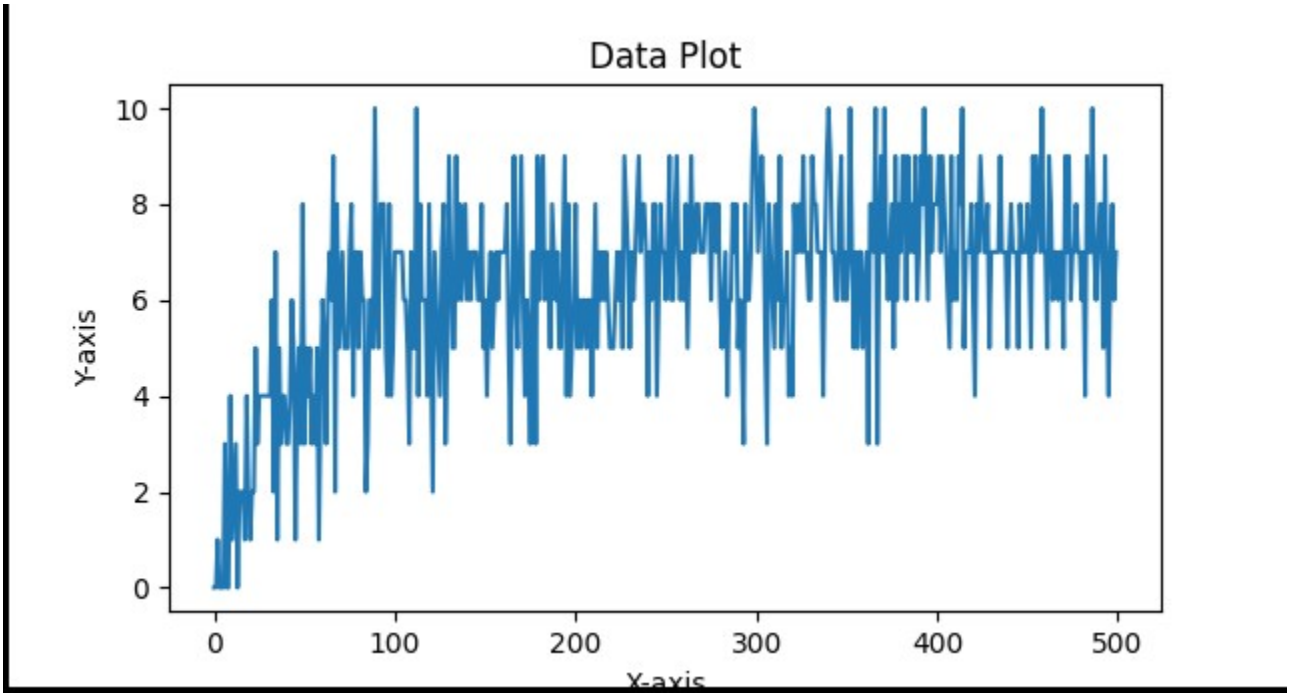
The pattern is clear the models for the math models seem to prefer few layers, many heads and more n_embd.

The performance still leaves something to be desired. This time to illustrate I ran the following command on the nohup.out file where the output from running the code went…

```
grep "samples that are in train" nohup.out | cut -c 1-2 > score.txt
```

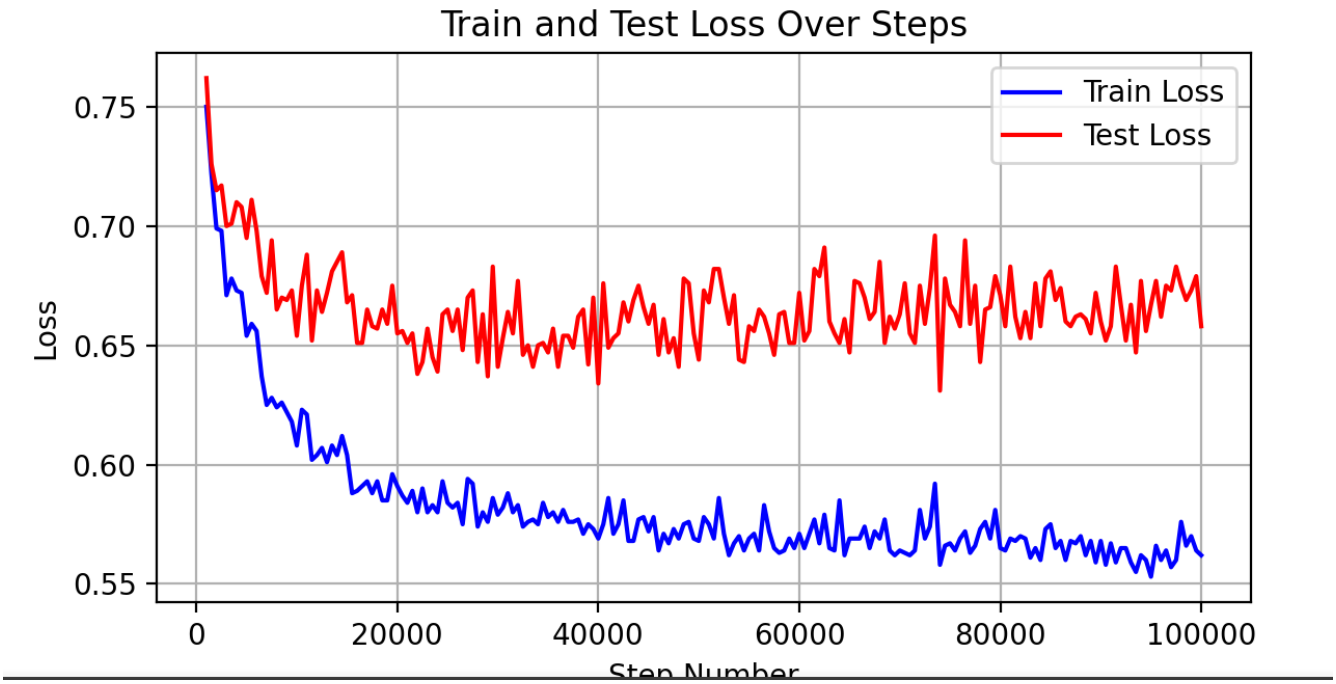
This gets the numbers that are in the training set, which as this dataset is a list of unique examples all are in the training set. So anything out of this result is wrong. A plot of score.txt illustrates how the performance improves and levels off. This is a plot of 100000 iterations, the score is report every 200 steps, hence the 500 units on the X axis.

Definitely room for improvement.



To be through here is the plot of the losses to see that the test loss flattens out and the train loss drops more but flattens as well, if anything it appears to rise ever so slightly. The fact that the test loss is more than the training loss suggests that the model is over-fitting. Probably cutting it off somewhere around 20000 steps would be good enough. Using option -e 1 to use an early stopping on the first instance of no reductions of loss, results in stopping at step 4000. Using -e 3, 13500 steps. -e 10 gives a reasonable 21500 steps.

Using plot-makemore-learning-curves.py allows plotting of the data from the file makemore_log.csv



Final Try for now

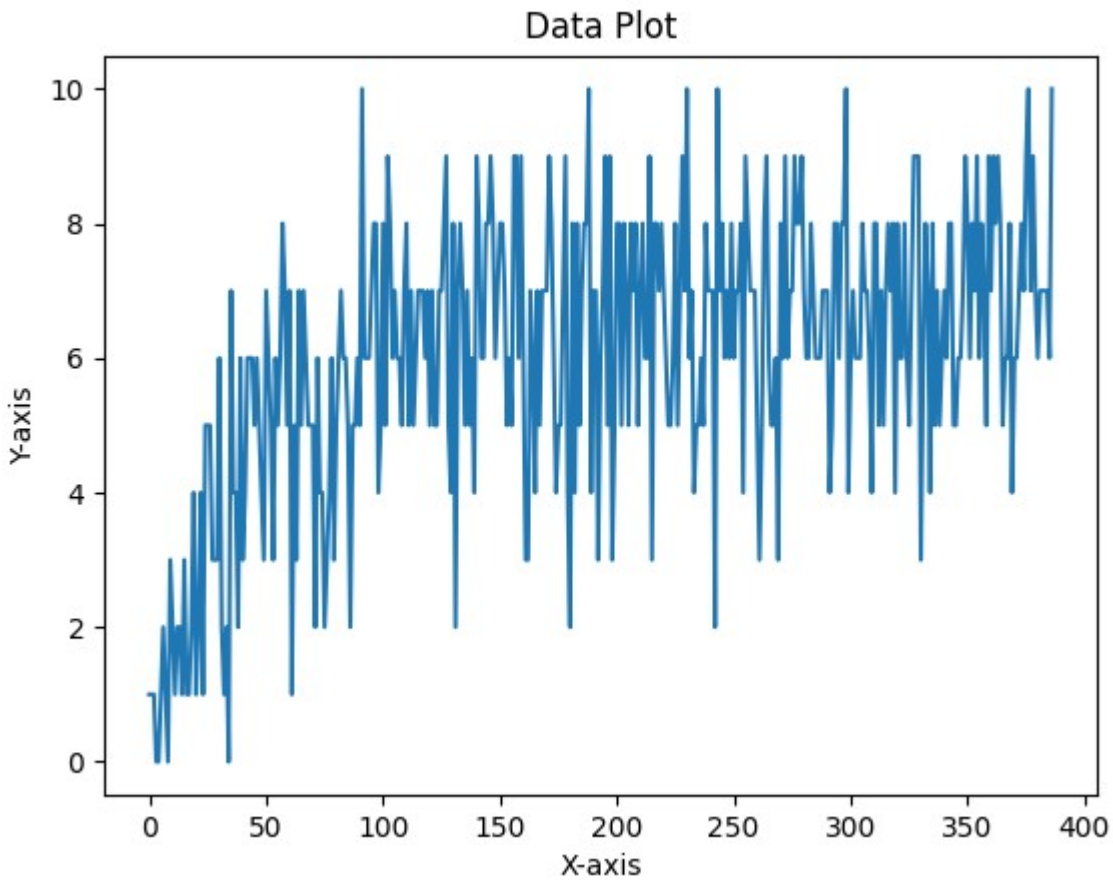
```
0x9e2d48 {'input_file': 'math_dataset_28_sorted.txt', 'work_dir': 'math-28', 're
sume': False, 'sample_only': False, 'num_workers': 4, 'max_steps': 100001, 'devi
ce': 'cpu', 'seed': 3407, 'top_k': -1, 'type': 'transformer', 'n_layer': 1, 'n_h
ead': 16, 'n_embd': 48, 'n_embd2': 32, 'batch_size': 32, 'learning_rate': 0.01,
'weight_decay': 0.01, 'lt_thr': 10000000000.0, 'early_stop': 100, 'max_params':
10000000000.0}
number of examples in the dataset: 3248
max word length: 16
number of unique characters in the vocabulary: 17
vocabulary:
 *+-./0123456789=
split up the dataset into 2924 training examples and 324 test examples
dataset determined that: vocab_size=18, block_size=17
number of parameters: 0.03M
model num. Params: 30912, model.pt size of 129735 bytes.
```

After another grid search allowing models up to 100000 parameters, we find one that has only 1 layer, 16 heads, 48 embd, lr 0.01.

Running with option -e 100 it will run to…

```
step 77500 train loss: 0.5636131763458252 test loss: 0.694291889667511
0.6692652702331543 is best loss at step 77500, Early Stopping Applied. tag:
0x9e2d48
```

Definitely overfitting this time.



Above plot, performance once again flattens out early on correct scores.

Appendix 3

Love quotes that are interesting from grepping on “Love” on the nohup.out file that was created while training makemore on the 10M quotes file, quotes_all.txt.csv…

They are chronological in the sense that the model was running and the ones on top are earlier in the run.

```
Love cool a feeling grow to the marriage, I started or touched do never that it's
also builded gramo.

Love most never when minds and and priority.

Love means fellow.

Love in my whole can't know, when you have one cooking has to have the education of
the songstering.

Love good all there world. I think wondendy is because I'd go to take one those
house hels when you that we more emotional effer his ains and get once the world it
was learning.
```

```
Love and myself about society, 'I want an act. It's to get make a little right
making and weekship and advanting to faware. My dad, try people! It is fascine
moved of their full new.

Love is being very body it. an inventions for the stuff. Hotf, time with a view
really force by Family eveign, the lively smetated to bond history by them it.

Love is in way the power opposed of the great sexy.

Love no diety is by the starter that may be believe and look.

Love men may be call fear of God from - also attack world.


Love serious capable education is the such skin.

Love is death one to as personality is the purpose. If it's hide of God, and as the
pity of thankful - that is its idiocally ultime, you started.

Lovers can prayer the sense of my uncertain. You should be an university and good
money Scor-Illay professioned. If teacher.

Love is one into God go to period environment at the family is those other
greatest.

Love to reform and unable full because, as blind presented for lead.

Love one's happiness, but made stressed it is to learn away that people thinking in
each of love by that.

Love is life is to become he hope, but in the heroism.

Love is the condition, every all men alone, we've great every third hope for it.

Love that stick would provide, more of money consider than the history of
respected, by his flowers him differents.
```

Appendix 4

The following was generated by training the distil-gpt2 model on the quotes csv file. In this case I used what I refer to as q-inference where the code creates 10 inferences and picks the best one. I plan on putting this code on GitHub when it is ready, TBD.

```
python3 q-inference.py quotes

Using model path: quotes
Prompt: The
0 -13.700541496276855
1 -26.85333251953125
2 -27.804710388183594
3 -31.28477668762207
4 -17.877981185913086
5 -23.032310485839844
6 -32.729827880859375
7 -24.180086135864258
8 -27.836820602416992
```

```
9 -24.09499740600586


The world is never divided into the bad and the good, while everyone in the rich is
in the good. Thus, the good and the bad are always a combination.
I think the idea that anyone can become the best doctor in the world is insane.
The good news is, that Barack Obama, a brilliant political speechwriter, is less of
an inspirational man than a charming, charming politician.
I believe that being the best husband in the world is the greatest gift.
An artist is never too old to paint what he does.
The thing that is called an artist is to know the best of what he is doing. That is
called an artist because he is always engaged in listening.
The greatest gift is the gift.
The best way to give is to make others to see who is giving.
Be honest, give what you think.
Be true to yourself, be good to yourself, make others know what you are. Be true to
yourself, make others great to yourself, make others proud, make your own
everything else look better, make yourself considerate, make everything else make
what you are. Be strong to yourself, make what you are. Be true to yourself try to
your best, make other people beautiful, make all else do it. Be true to yourself
good to yourself, make others feel that you would be the best, make yourself so
that you are. Be true to yourself a masterpiece.


Time: 37.74
```

The model was trained for 51 epochs, 1 to test and see timing, 10 for another test overnight and finally 40 more to bring the loss down to under 2…

{'loss': 1.9966, 'learning_rate': 8.496176720475786e-08, 'epoch': 39.93}

The quotes are better than seen by using makemore. I am thinking some of this is part of the fact that it is a transfer learn situation rather than from scratch. I am also thinking it has partly to do with the fact that sub-word tokens are used versus character tokens. It is possible that the standard GPT2 learning gets a boost by ‘knowing’ parts of words already. I plan on exploring this a bit more, maybe by erasing the distil-gpt-2 model by training it on high entropy white noise type of data and training a character level GPT2 with the same parameters as the standard GPT2 via nanoGPT….TBD.

A few more that I cherry picked that were interesting….

T*he secret of being great is not finding a perfect balance between effort and failure.*

The past is a metaphor for the future. We do not need to reconstruct our past.

The future is never a destination it is a destiny it is a vision, a way of living and dreaming.
You do not stop dreaming tomorrow because the past is only dream.
We are in the present moment.
Bad government is good for bad people.

Prompted with Happiness…
Happiness lies in the ways we seek, not in ourselves.

You must love yourself, not in everything.
Happiness lies in our way of knowing and loving ourselves.
Happiness lies in what is easy to find.
Happiness lies in the ways we must seek.
Happiness lies in our way of being.
Happiness lies in giving, not in our ways of being too selfish.
Happiness lies in the ways we find, not in our ways.
Happiness is not how much trouble we get, but in our way.
Happiness lies in giving.

Finally, try something not in the quote dataset, which does not contain the word Misato. A homage to , which helped me get motivated again with ML projects, especially the quotes training as a warm-up for something bigger in the future.

Using model path: quotes
Prompt: Misato wrote this on the blackboard,

Misato wrote this on the blackboard, and I'd like to get behind computers and get behind the wheel of this car.
I'm very proud.
I'm not afraid of computers, which I think they are the devil. But if they're not devil, I can't imagine they would be.
There are too many computers today that do not ask for much.
The internet is not just for those who've put up with their own little wheelie machines. They are all for those who've been using computers for a bit of creative fun.
The reason that most people don't take computers off on computers before they set up a movie or internet is that they don't take them out of their house when they go to bed or have tea in the morning. They don't take their kids to school or work.
I don't play Xbox. I can take care of my car and watch my candles and my candles, I'm not worried about computers at all. I'm just happy playing Apple TV.
I would love to come in and do anything on computers with my iPhone.
Internet research and other things. I'm not sure about computers. I'm into the surround myself with artificial intelligence, whether it's a machine will ever be my main concern. The driving machine will be the computer and whether it is going to be the computer, when I can come in and going.

Time: 11.83