# wM4 Lab: Language Modeling

## Introduction and setup

In this lab, we will practice the main techniques for building N-gram based language models for our speech recognizer. In subsequent modules, the resulting LM will be used in the speech recognition decoder, together with the acoustic model from the preceding lab.

This lab is carried out in a Linux command shell environment. The get started, make sure you know how to invoke the Linux [bash](http://man7.org/linux/man-pages/man1/bash.1.html) shell, either on a native Linux system, using Cygwin on a Windows system, or in the Windows subsystem for Linux. Bash is the default shell on most systems.

Inside bash, change into the M4\_Language\_Modeling directory:

* cd M4\_Language\_Modeling

We will be using pre-built executables from the SRI Language Modeling toolkit ([SRILM](http://www.speech.sri.com/projects/srilm/)). Start by adding the SRILM binary directories to your search path. If you are in a Cygwin,

* PATH=$PWD/srilm/bin/cygwin64:$PWD/srilm/bin:$PATH

In Linux or Windows Subsystem for Linux, use

* PATH=$PWD/srilm/bin/i686-m64:$PWD/srilm/bin:$PATH

You can put this command in the .bashrc file in your home directory, so it is run automatically next time you invoke the shell. Also, make sure you have the [gawk](http://man7.org/linux/man-pages/man1/gawk.1.html) utility installed on your system. As a check that all is set up, run

* ngram-count -write-vocab –
* compute-oov-rate < /dev/null

which should each output a few lines of text without error messages. It will be helpful to also install the optional [wget](http://man7.org/linux/man-pages/man1/wget.1.html) command.

Since language modeling involves a fair amount of text processing it will be useful to have some familiarity with Linux text utilities such as [sort](http://man7.org/linux/man-pages/man1/sort.1.html), [head](http://man7.org/linux/man-pages/man1/head.1.html), [wc](http://man7.org/linux/man-pages/man1/wc.1.html), [sed](http://man7.org/linux/man-pages/man1/sed.1.html), [gawk](http://man7.org/linux/man-pages/man1/gawk.1.html) or [per](https://perldoc.perl.org/perl.html)l, and others, as well as Linux mechanisms for redirecting command [standard input/output](https://en.wikipedia.org/wiki/Standard_streams), and [pipelining](https://en.wikipedia.org/wiki/Pipeline_%28Unix%29) several commands. We will show you commands that you can copy into the shell to follow along, using this symbol

* command argument1 argument2 …

but we encourage you try your own solutions to achieve the stated goals of each exercise, and to explore variants.

## Preparing the data

We will be using the transcripts of the acoustic development and test data as our dev and test sets for language modeling.

**TASK:** Locate files in the ‘data’ subdirectory and count the number of lines and words in them.

***SOLUTION:***

* ls data
* wc -wl data/dev.txt data/test.txt

**TASK:** View the contents of these files, using your favorite pager, editor, or other tool. What do you notice about the format of these files? How do they differ from text you are used to?

**SOLUTION**

* head data/\*.txt

You will notice that the data is in all-lowercase, without any punctuation. This is because we will model sequences of words only, devoid of textual layout, similar to how one your read or speak them. The spelling has to match the way words are represented in the acoustic model. The process of mapping text to the standard form adopted for modeling purposes is called text normalization (or TN for short), and typically involves stripping punctuation, mapping case, fixing typos, and standardizing spellings of words (like MR. versus MISTER). This step can consume considerable time and often relies on powerful text processing tools like sed or perl.

Because it is so dependent on the source of the data, domain conventions, and tool knowledge, we will not elaborate on it here. Instead, we will download an LM training corpus that has already been normalized,

* wget http://www.openslr.org/resources/11/librispeech-lm-norm.txt.gz

If your system doesn’t have the wget command you can download this file in a browser and move it into the LM lab directory.

**TASK:** Inspect the file and count lines and word tokens. How does the text normalization of this file differ from our test data?

***SOLUTION:***  The file is compressed in the gzip (.gz) so we must use the [gunzip](https://www.gnu.org/software/gzip/manual/gzip.html) tool

* gunzip -c librispeech-lm-norm.txt.gz | head
* gunzip -c librispeech-lm-norm.txt.gz | wc -wl

The second command can take a while as the file is large. You will notice that this file is text normalized but uses all-uppercase instead of all-lowercase.

Language model training data, and language models themselves, are often quite large but compress well since they contain text. Therefore, we like to keep them in compressed form. The SRILM tools know how to read/write .gz files, and it is easy to combine gzip/gunzip with Linux text processing tools.

***OPTIONAL TASK:*** *Download the raw training data at* [*http://www.openslr.org/12/librispeech-lm-corpus.tgz*](http://www.openslr.org/12/librispeech-lm-corpus.tgz)*, and compare it to the normalized text. How would you perform TN for this data?*

***SOLUTION*** left to the reader!

## Defining a vocabulary

The first step in building a LM is to define the set of words that it should model. We want to cover the largest possible share of the word tokens with the smallest set of words, so as to keep model size to a minimum. That suggests picking the words that are most frequent based on the training data.

One of the functions of the [ngram-count](http://www.speech.sri.com/projects/srilm/manpages/ngram-count.1.html) tool is to count word and ngram occurrences in a text file.

* ngram-count -text TEXT -order 1 -write COUNTS -tolower

Will count 1-grams (i.e., words) and write the counts to a file. The final option above maps all text to lowercase, thus dealing with the mismatch we have between our training and test data.

**TASK:** Extract the list of the 10,000 most frequent word types in the training data. What kinds of words do you expect to be at the top of the list? Check your intuition.

**HINT:** Check out the Linux sort, head, and cut commands.

***SOLUTION:***

* ngram-count -text librispeech-lm-norm.txt.gz -order 1 -write librispeech.1grams -tolower
* sort -k 2,2 -n -r librispeech.1grams | head -10000 > librispeech.top10k.1grams
* cut -f 1 librispeech.top10k.1grams | sort > librispeech.top10k.vocab

The intermediate file librispeech.top10k.1grams contains the words and their counts sorted most frequent first. As you might expect, common function words liked “the”, “and”, “of” appear at the top of the list. Near the top we also find two special tags, <s> and </s>. These are added by ngram-count to mark the start and end, respectively, of each sentence. Their count equals the number of non-empty lines in the training data, since it is assumed that each line contains one sentence (empty lines are ignored).

We now want to find out how well out 10k vocabulary covers the test data. We could again use Linux tools for that, but SRILM contains a handy script [compute-oov-rate](http://www.speech.sri.com/projects/srilm/manpages/training-scripts.1.html) that takes two arguments: the unigram count file and the list of vocabulary words.

**TASK:** What is the rate of out-of-vocabulary (OOV) words on the training, dev and test sets?

**HINT:** Use the same method as before to generate the unigrams for dev and test data.

**SOLUTION:**

* compute-oov-rate librispeech.top10k.vocab
* ngram-count -text data/dev.txt -order 1 -write dev.1grams
* compute-oov-rate librispeech.top10k.vocab dev.1grams
* ngram-count -text data/test.txt -order 1 -write test.1grams
* compute-oov-rate librispeech.top10k.vocab test.1grams

Usually we expect the OOV rate to be lowest on the training set because we used it to select the words (the vocabulary is biased toward the training set), but in this case the test sets have been chosen to be “cleaner” and have lower OOV rates. (The training data actually contains some languages other than English, though most of those will not make it into the vocabulary.)

Note that compute-oov-rate also reports about “OOV types”. OOV types are the number of unique words that are missing from the vocabulary, regardless of how many times they occur.

The OOV rate of around 5% is quite high – remember that we will never be able to recognize those OOV words since the LM does not include them (they effectively have probability zero). However, we chose the relatively small vocabulary size of 10k to speed up experiments with the decoder later.

**OPTIONAL TASK:** Repeat the steps above for different vocabulary sizes (5k, 20k, 50k, 100k, 200k). Plot the OOV rate as a function of vocabulary size. What shape do you see?

## Training a model

We are now ready to build a language model from the training data and the chosen vocabulary. This is also done using the ngram-count command. For instructional purpose we will do this in two steps: compute the N-gram statistics (counts), and then estimate the model parameters. (ngram-count can do both in one step, but that’s not helpful to understand what happens under the hood.)

***TASK:*** *Generate a file containing counts of all trigrams from the training data. Inspect the resulting file*

**HINT:** Consult the [ngram-count](http://www.speech.sri.com/projects/srilm/manpages/ngram-count.1.html) man page and look up the options -order, -text, and -write. Remember the case mismatch issue.

**SOLUTION: The first command uses about 10GB of memory and takes 15 minutes on a 2.4GHz Intel Xeon E5 CPU, so be sure to procure a sufficiently equipped machine and some patience.**

* ngram-count -text librispeech-lm-norm.txt.gz -tolower -order 3 -write librispeech.3grams.gz
* gunzip -c librispeech.3grams.gz | less

Note that we want to compress the output file since it is large. The -order option in this case is strictly speaking optional since order 3 is the default setting. Note that the output is grouped by common prefixes of N-grams, but that the words themselves are not alphabetically sorted. You can use the -sort option to achieve the latter.

Now we can build the LM itself. (Modify the output file names from previous steps according to your own choices.)

***TASK:*** *Estimate a backoff trigram LM from librispeech.3grams.gz, using the Witten-Bell smoothing method.*

**HINT:** Consult the [ngram-count](http://www.speech.sri.com/projects/srilm/manpages/ngram-count.1.html) man page for options -read, -lm, -vocab, and -wbdiscount .

**SOLUTION:**

* ngram-count -debug 1 -order 3 -vocab librispeech.top10k.vocab -read librispeech.3grams.gz -wbdiscount -lm librispeech.3bo.gz

We added the -debug 1 option to output a bit of information about the estimation and resulting LM, in particular the number of N-grams output.

We will now try to understand the way LM parameters are stored in the model file. Peruse the file using

* gunzip -c librispeech.3bo.gz | less

or, if you prefer, gunzip the entire file using

* gunzip librispeech.3bo.gz

and open librispeech.3bo in an editor. Note: the editor better be able to handle very large files -- the LM file has a size of 1.3 GB.

# Model evaluation

Consult the description of the backoff LM file format [ngram-format(5)](http://www.speech.sri.com/projects/srilm/manpages/ngram-format.5.html), and compare to what you see in our model file, to be used in the next task.

***TASK:*** *Given the sentence “a model was born”, what is the conditional probability of “born”?*

**SOLUTION:** The model is a trigram, so the longest N-gram that would yield a probability to predict “born” would be “model was born”. So let’s check the model for that trigram. (One way to locate information in the model file is the [zgrep](https://linux.die.net/man/1/zgrep) command, which searches a compressed file for text strings. *Each search string below starts with a TAB character* to avoid spurious matches against other words that contain the string as a suffix. You can use your favorite tools to perform these searches.)

* zgrep " model was born" librispeech.3bo.gz

This outputs nothing, meaning that trigram is not found in the model, and we have to use the back-off mechanism. We look for the line that contains the context bigram “model was” following a whitespace character:

* zgrep -E "\smodel was" librispeech.3bo.gz | head -1

-2.001953 model was 0.02913048

The first number is the log probability P(was | model), which is of no use to use here. The number at the end is the backoff weight associated with the context “model was”. It, too, is encoded as a base-10 logarithm. Next, we need to find the bigram probability we’re backing off to, i.e., P(born | was):

* zgrep -E “\swas born" librispeech.3bo.gz | head -1

-2.597636 was born -0.4911189

The first number is the bigram probability P(born | was). We can now compute the log probability for P(born | model was) as the sum of the backoff weight and the bigram probability:

0.02913048 + -2.597636 = -2.568506, or as a linear probability 10-2.568506 = 0.002700813.

***TASK:***  *Compute the total sentence probability of “a model was born” using the ngram -ppl function. Verify that the conditional probability for “born” is as computed above.*

**SOLUTION:** We feed the input sentence to the [ngram](http://www.speech.sri.com/projects/srilm/manpages/ngram.1.html) command in a line of standard input, i.e., using “-“ as the filename argument to -ppl. Use the option -debug 2 to get a detailed breakdown of the sentence-level probability:

* echo "a model was born " | ngram -debug 2 -lm librispeech.3bo.gz -ppl –

a model was born

p( a | <s> ) = [2gram] 0.01653415 [ -1.781618 ]

p( model | a ...) = [3gram] 0.0001548981 [ -3.809954 ]

p( was | model ...) = [3gram] 0.002774693 [ -2.556785 ]

p( born | was ...) = [2gram] 0.002700813 [ -2.568506 ]

p( </s> | born ...) = [3gram] 0.1352684 [ -0.8688038 ]

1 sentences, 4 words, 0 OOVs

0 zeroprobs, logprob= -11.58567 ppl= 207.555 ppl1= 787.8011

Notice how ngram adds the sentence start and end tags, <s> and </s>. The final line gives both the log probability and the perplexity of the entire sentence. The line starting “p(born | was …)” has the conditional word probability that we computed previously. The label “[2gram]” indicates that a backoff to bigram was used. The final “logprob” value -11.58567 is just the sum of the log probabilities printed for each word token. Let’s verify the perplexity value based on it’s definition: we divide the logprob by the number of word tokens (including the end-of-sentence), convert to a probability and take the reciprocal (by negating the exponent): 10- (-11.58567 / 5) = 207.555. Of course this is not a good estimate of perplexity as it is based on only 5 data points.

***TASK:*** *Compute the perplexity of the model over the entire dev set.*

**SOLUTION:** The exact same invocation of ngram can be used, except we use the file containing the dev set as ppl input. We also omit the -debug option to avoid voluminous output. Note: these commands take a few seconds to run, only because loading the large LM file into memory takes some time – the model evaluation itself is virtually instantaneous.

* ngram -lm librispeech.3bo.gz -ppl data/dev.txt

file dev.txt: 466 sentences, 10841 words, 625 OOVs

0 zeroprobs, logprob= -21939 ppl= 113.1955 ppl1= 140.4475

We thus have a perplexity of about 113. The first line of summary statistics also gives the number of out-of-vocabulary words (which don’t count toward the perplexity, since they get probability zero). In this case the OOV rate is 625/10841 = 5.8%.

Running the same command on the test set (data/test.txt) yields a perplexity of 101 and an OOV rate of 4.9%. Both statistics indicate that the test portion of the data is a slightly better match to our model than the development set.

**TASK:** Vary the size of the training data and observe the effect this has on model size and quality (perplexity).

**HINT:** The original librispeech-lm-norm.txt.gz has about 40 million lines. Use gunzip and the [head](http://man7.org/linux/man-pages/man1/head.1.html) command to prepare training data that is ½, ¼, …, of the full size. (This is very easy, but can you think of better ways to pare down the data?)

**SOLUTION:** Rebuild the model (*using the original vocabulary*), and evaluate perplexity for different amounts of data. Plot model size (number of ngrams in the head of the model file) and perplexity as a function of training data size. Details left to the student, using the steps discussed earlier.

# Model adaptation

We will now work through the steps involved in adapting an existing LM to a new application domain. In this scenario we typically have a small amount of training data for the new, target domain, but a large amount, albeit mismatched data from other sources. For this exercise we target the [AMI domain](http://www.amiproject.org/) of multi-person meetings as our target domain. The language in this are spontaneous utterances from face-to-face interactions, whereas the “librispeech” data we used so far consisted of read books, a dramatic mismatch in speaking styles and topics.

We will use the “librispeech” corpus as our out-of-domain data, and adapt the model we just created from that corpus to the AMI domain, using a small amount of target-domain data corpus. Corpus subsets for training and test are in the data directory:

* wc -wl data/ami-\*.txt

1. 6473 data/ami-dev.txt

2096 20613 data/ami-test.txt

86685 924896 data/ami-train.txt

Also provided is a target domain vocabulary consisting of all words occurring at least 3 times in the training data, consisting of 6171 words:

* wc -l data/ami-train.min3.vocab

6271 data/ami-train.min3.vocab

***TASK:*** *Build the same kind of Witten-Bell-smoothed trigram model as before, using the provide AMI training data and vocabulary. Evaluate its perplexity on the AMI dev data.*

**SOLUTION:**

* ngram-count -text data/ami-train.txt -tolower -order 3 -write ami.3grams.gz
* ngram-count -debug 1 -order 3 -vocab data/ami-train.min3.vocab -read ami.3grams.gz -wbdiscount -lm ami.3bo.gz
* ngram -lm ami.3bo.gz -ppl data/ami-dev.txt

file data/ami-dev.txt: 2314 sentences, 26473 words, 1264 OOVs

0 zeroprobs, logprob= -55254.39 ppl= 101.7587 ppl1= 155.5435

***TASK:*** *Evaluate the previously built librispeech model on the AMI dev set.*

**SOLUTION:** Again, this takes a few seconds due to the loading time of the large model.

* ngram -lm librispeech.3bo.gz -ppl data/ami-dev.txt

file data/ami-dev.txt: 2314 sentences, 26473 words, 3790 OOVs

0 zeroprobs, logprob= -56364.05 ppl= 179.8177 ppl1= 305.3926

Note how both the perplexity and the OOV count are substantially higher for this large model than for the much small, but well-matched AMI language model. If we modified the vocabulary of the old model to match the new domain its perplexity would increase further. (Can you explain why?)

We will now adapt the old model by interpolating it with the small AMI LM. As explained in the course materials, model interpolation means that all N-gram probabilities are replaced by weighted averages of the two input models. So we need to specify the relative weights of the two existing models, which must sum to 1. A good rule of thumb is to give the majority of weight (0.8 or 0.9) to the in-domain model, leaving a small residual weight (0.2 or 0.1) to the out-of-domain model.

***TASK:*** *Construct an interpolated model based on the existing librispeech and AMI models, giving weight 0.8 to the AMI model, and evaluate it on the dev set.*

**HINT:** Make use of the [ngram](http://www.speech.sri.com/projects/srilm/manpages/ngram.1.html) options -mix-lm, -lambda, and -write-lm.

* ngram -debug 1 -order 3 -lm ami.3bo.gz -lambda 0.8 -mix-lm librispeech.3bo.gz -write-lm ami+librispeech.bo.gz
* ngram -lm ami+librispeech.3bo.gz -ppl data/ami-dev.txt

file ami-dev.txt: 2314 sentences, 26473 words, 783 OOVs

0 zeroprobs, logprob= -56313.77 ppl= 102.546 ppl1= 155.6145

At first sight, this result is disappointing. Note how the perplexity is now 102, slightly up from the value that the AMI-only model produced. But also note how the number of OOVs was almost halved (from 1264 to 783), due to the addition of words covered by the out-of-domain model that were not in the AMI model. The model now has more words to choose from when making its predictions. An important lesson from this exercise is that we can only compare perplexity values when the underlying vocabularies are the same. Otherwise, the enlarged vocabulary is a good thing, as it reduces OOVs. The interpolation step has effectively adapted not only the model probabilities, but the vocabulary as well.

Still, it would be nice to do an apples-to-apples comparison to see the effect of just the probability interpolation on model perplexity. We can do this by telling the ngram tool to only use words from the AMI vocabulary in the interpolated model:

* ngram -debug 1 -order 3 -lm ami.3bo.gz -lambda 0.8 -mix-lm librispeech.3bo.gz -write-lm ami+librispeech.bo.gz -vocab data/ami-train.min3.vocab -limit-vocab

This is the same command as before, but with the -limit-vocab option added, telling ngram to only use the vocabulary specified by the -vocab option argument. We can now evaluate perplexity again:

file ami-dev.txt: 2314 sentences, 26473 words, 1264 OOVs

0 zeroprobs, logprob= -53856.04 ppl= 90.52426 ppl1= 136.8931

The number of OOVs is now back to the same as with ami.3bo.gz, but perplexity is reduced from 102 to 90.

***TASK (optional):*** *Repeat this process for different interpolation weights, and see if you can reduce perplexity further. Check results on both AMI dev and test sets.*

This step is best carried out using the enlarged vocabulary, since that is what we want to use in our final model. But notice how we are now effectively using the dev set to train another model parameter, the interpolation weight. The result will thus be tuned to the dev set. This is why we better have another test set held out (data/ami-test.txt in this case) to verify that the result of this tuning also improves the model (lowers the perplexity) on independent data.

The tuning of interpolation weights would be rather tedious if carried out by trial and error. Fortunately, there is an efficient algorithm that finds optimal weights based on [expectation maximation](https://en.wikipedia.org/wiki/Expectation%E2%80%93maximization_algorithm), implemented in the command compute-best-mix, described under [ppl-scripts](http://www.speech.sri.com/projects/srilm/manpages/ppl-scripts.1.html).

***TASK (optional):*** *Use compute-best-mix to find the best -lambda value for interpolation for the two models we built.*

**HINT:** As input to the command, generate detailed perplexity output for both models, using ngram -debug 2 -ppl data/ami-dev.txt .

# Model pruning

We saw earlier that model size (and perplexity) varies with the amount of training data. However, if a model gets too big for deployment as the data size increases it would be a shame to have to not use it just for that reason. A better approach is to train a model on all available data, and then eliminate parameters that are redundant or have little effect on model performance. This is what model pruning does.

A widely used [algorithm for model pruning](https://arxiv.org/pdf/cs/0006025v1.pdf) based on entropy is implemented in the ngram tool. The option -prune takes a small value, such as 10-8 or 10-9, and remove all ngrams from the model that (by themselves) raise the perplexity of the model less than that value in relative terms.

***TASK:*** *Shrink the large librispeech model trained earlier, using pruning values between 10-5 and 10-10 (stepping by powers of ten). Observe/plot the resulting model sizes and perplexities, and compare to the original model.*

**SOLUTION:** Starting with 1e-5 (= 10-5 in floating point notation), create the pruned model:

* ngram -debug 1 -lm librispeech.3bo.gz -prune 1e-5 -write-lm librispeech-pruned.3bo.gz

Then evaluate the librispeech-pruned.3bo.gz model on the entire dev set, as before. The -debug option lets the tool output the number of ngrams pruned and written out. Add the resulting number of bigrams and trigrams to characterize the pruned model size (roughly, the number of model parameters, since the number of unigrams is fixed to the vocabulary, and the backoff weights are determined by the probability parameters).

# End of M4 Lab

Congratulations!