MPhil in Machine Learning and Machine Intelligence

Module MLMI2: Speech Recognition

L7: N-Gram Language Models

Phil Woodland
pcw@eng.cam.ac.uk

Michaelmas 2021

Cambridge University Engineering Department

Introduction

In order to use Bayes' decision rule need to be able to have the prior of a class

- often in speech and language processing this is the **sentence probability** $P(\boldsymbol{W})$
- for speech recognition need to combine acoustic "scores" with this prior

$$\hat{\boldsymbol{W}} = \underset{\boldsymbol{W}}{\text{argmax}} \{P(\boldsymbol{W}|\boldsymbol{O})\} = \underset{\boldsymbol{W}}{\text{argmax}} \{p(\boldsymbol{O}|\boldsymbol{W})P(\boldsymbol{W})\}$$

The sentences can be written as a joint probability of all the words in the sentence:

$$P(\boldsymbol{W}) \quad = \quad P(w_0, w_1, \ldots w_K, w_{K+1})$$

In this lecture we will

- Introduce the language modelling problem, and LM assessment
- Describe ***N*-gram** language models
- **Mixture** language models
- **Class-based** *N*-gram language models

Later look at **neural network LMs** which are often used in combination with word *N*-grams (especially when used in **language model re-scoring**, which is also discussed later).

Language models for ASR

The language model needs to compute the probability of any word string, $P(\boldsymbol{W})$.

Overall string probability can be decomposed into **word prediction probabilities** given a **history**.

First sentence start, <s>, and sentence end markers, </s>, are added to each sentence.

<s>	THE	OVERALL	OPERATION	...	STRING	</s>
w_0	w_1	w_2	w_3	...	w_K	w_{K+1}

w_0 is the sentence start marker and w_{K+1} is the sentence end marker.

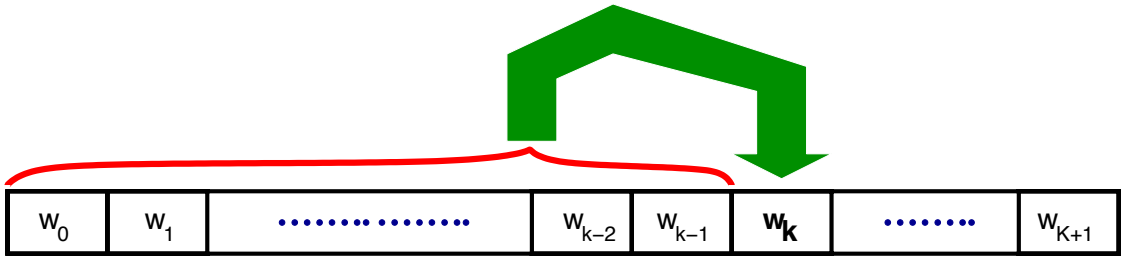
The probability of a K word sentence $\boldsymbol{W}$ $(= w_1 \dots w_K)$ can be computed by

$$P(\boldsymbol{W}) = \prod_{k=1}^{K+1} P\left(w_k | w_0 \dots w_{k-2} w_{k-1}\right)$$

where w_0 is the sentence start marker and w_{K+1} is the sentence end marker. Note that this expression includes the probability of the sentence end marker.

LMs for ASR - Prediction

If the search is to be performed from left-to-right through the utterance a **decomposition** of $P(\mathbf{W})$ in such a form is **required** so that the language model allows the **prediction** of the next word, w_k in the sequence from the current history.



Therefore language models for recognition should admit this decomposition and, if they are to be used **directly during search**, are required to be predictive. The use of a sentence end marker allows the prediction of the end of a sentence.

These word-by-word conditional probabilities can be used to set the word-to-word transition probabilities in a network encoding the set of allowed sentences (often all word sequences) and are used directly in the search to find the most likely word (sequences).

N-Gram Language Models

Consider a task with a **vocabulary** of V words (LVCSR 65k+)

- 10-word sentences yield (in theory) V^{10} probabilities to compute
- not every sequence is valid but number still vast for LVCSR systems

Need to partition histories into appropriate equivalence classes

Approximation: all histories with the same first word have the same probability

$$P(\text{bank}|\text{I}, \text{robbed}, \text{the}) \approx P(\text{bank}|\text{I}, \text{fished}, \text{from}, \text{the}) \approx P(\text{bank}|\text{the})$$

- simple form of equivalence mappings - **bigram** language model

$$P(\boldsymbol{W}) = \prod_{k=1}^{K+1} P(w_k|w_0, \ldots w_{k-2}, w_{k-1}) \approx \prod_{k=1}^{K+1} P(w_k|w_{k-1})$$

N-Gram Language Models

The simple bigram can be extended to general N-grams

$$P(\boldsymbol{W}) = \prod_{k=1}^{K+1} P(w_k|w_0, \ldots w_{k-2}, w_{k-1}) \approx \prod_{k=1}^{K+1} P(w_k|w_{k-N+1}, \ldots, w_{k-1})$$

In an N-gram, **histories equivalent if most recent $N-1$ words are the same**.

Number of model parameters scales with the size of N (consider $V = 65\text{k}$):

- N=1: **unigram**: $65\text{k}^1 = 6.5 \times 10^4$
- N=2 **bigram**: $65\text{k}^2 = 4.225 \times 10^9$
- N=3 **trigram**: $65\text{k}^3 = 2.746 \times 10^{14}$
- N=4: **4-gram (quadrigram)**: $65\text{k}^4 = 1.785 \times 10^{19}$

Web has 100's of billions of pages indexed by Google - not enough data!

- Longer-span models should be more accurate, but large numbers of parameters

A central problem is how to get robust estimates with a long-span?

Note that N-gram probability estimates (and word frequencies) depend crucially on the source of the text data used to train them: poetry vs financial news ....

Lexical coverage

Another important issue is **lexical coverage**.

For any LM for speech recognition we want to minimise the number of out-of-vocabulary (**OOV**) words.

- In order to train language models each word has to occur sufficiently often in the training data.
- For each OOV word a speech recogniser using an *N*-gram LM typically makes 1.6 word errors.

For English business newspaper text a

- 5k vocabulary would typically has a 9% OOV rate;
- 20k 2% OOV rate
- 65k 0.6% OOV rate

Higher lexical coverage at a particular vocabulary size can be obtained if vocabulary is tailored for a particular individual or topic.

Note that **sub-word** language models can also be used and if **word-pieces** including **characters** are used as sub-words then there might be no (or far-fewer) OOVs, but still need to have examples to get good estimates and be able to cope with (much) longer spans.

Language Dependent Issues

Much work on language modelling (& speech recognition in general) has been done using English and views the **basic units** as “words” i.e. sequences of characters delimited by white space (or punctuation).

However some languages have rather different properties as far as language modelling is concerned.

- Some languages such as Chinese are written without spaces between words: need to first find word boundaries.
- German which allows widespread **compounding** and hence to have the same lexical coverage needs a much higher vocabulary size.
- In **highly inflected languages** (such as Russian or Turkish) which for a similar lexical coverage requires an order of magnitude higher vocabulary size (e.g. 500k words for 1% OOV rate on general Russian text). Furthermore such languages often have a much freer word order than that allowed in English.

These final two points lead to (at least) two problems if word modelling is used: increased **data sparsity** in estimating LM parameters; and a requirement for a recogniser that can handle more than 65,535 words (a common limit due to the desire to represent a word as a 2 byte quantity).

Assessing Language Models

Best way of assessing language models is the task (e.g. speech recognition, machine translation) performance criterion

- for speech recognition build acoustic and separate language models
- perform decoding and rank LMs according to word error rate

However, there are a number of problems

- can be expensive, to test with all LMs
- want to be able to test LMs independent of other models in the system
- need to be able to determine component of performance solely due to language model for each situation

Most commonly used way of independently assessing LMs is **perplexity**

- may be viewed as the average branching factor
- equivalent size of vocabulary if all LM probabilities the same

Entropy and Perplexity

Entropy is a measure of information - it can be used to show

- how much information is in a particular language model
- how well a given LM matches a certain task
- how well a word predicts the next word

The **entropy**, H, and **perplexity**, PP, of a discrete random event, W, given by

$$H = -\sum_{W \in \mathcal{V}} P(W) \log_2(P(W)), \quad PP = 2^H$$

where $\mathcal{V}$ is the set of all possible events

Interested in perplexity for language modelling

- W is a **word** (possibly given a history)
- $\mathcal{V}$ is the **vocabulary** (including start and end tokens)

The highest perplexity value for a particular vocab size is obtained in case of a uniform distribution: identical to the overall number of words

In general perplexity describes the equivalent size of the vocabulary if all words (sequences) were equally probable.

- Hence it can be viewed as equivalent to the **average branching factor** of the grammar.

Test-set perplexity

The definition of perplexity in the above form is:

- difficult to evaluate when incorporating word history into LMs
- relies that the language model probability estimates are accurate!
- not useful to assess how well specific text is modelled with a given LM

Quality of a LM is usually measures by the **test-set perplexity**

- compute the **logprob** (*LP*) i.e. average log probability per word

$$LP = \lim_{K\to\infty} -\frac{1}{K+1}\sum_{k=1}^{K+1} \log_2 P\left(w_k | w_0 \dots w_{k-2} w_{k-1}\right)$$

In practice *LP* must be estimated from a (finite-sized) portion of test text

- This is a (finite-set) estimate of the entropy
- Test-set perplexity, *PP*, can be found as $PP = 2^{LP}$

From the above the definition of PP can also be given as

$$\mathrm{PP} = \lim_{K\to\infty} \left\{ P(w_0, w_1, \dots, w_{K+1})^{-\frac{1}{K+1}} \right\}$$

or the reciprocal of the geometric mean of the per word probabilities.
These expressions include the prediction of `<SENT_END>`, but not `<SENT_START>` (since this always occurs at the start of sentences).

Relationship between PP and WER

In general as the perplexity decreases so does the **word error rate** (WER).

- **BUT** mainly very bad LM predictions lead to LM "generated" word errors
- possible to improve "good" predictions without decreasing WER

For the same general type of language model (e.g. *N*-gram):

- Usually fairly consistent relationship between test-set *PP* and WER
- A "rule-of-thumb" states that

  $$\mathrm{WER} = k \times \sqrt{PP}$$

  where *k* is a task dependent constant.
- Allows a prediction of the change in WER for a change in perplexity

N-Gram Language Model Estimation

Basic idea is to use **relative frequency** to estimate probabilities.

To estimate the probability of a particular trigram it is necessary to find the "count" (frequency of occurrence) of triple $\omega_i, \omega_j, \omega_k$.

$$\hat{P}(\omega_k|\omega_i,\omega_j) = \frac{f(\omega_i,\omega_j,\omega_k)}{\sum_{k=1}^{V} f(\omega_i,\omega_j,\omega_k)} = \frac{f(\omega_i,\omega_j,\omega_k)}{f(\omega_i,\omega_j)}$$

where $f(a,b,c,\ldots)$ = number of times that the word sequence (*event*) "a b c ..." occurs in the training data, V is the number of words in the vocabulary.

This is the **maximum likelihood** estimate

- excellent model of the training data ...
- many possible events will not be seen, zero counts - zero probability
- rare events, $f(\omega_i,\omega_j)$ is small, estimates unreliable

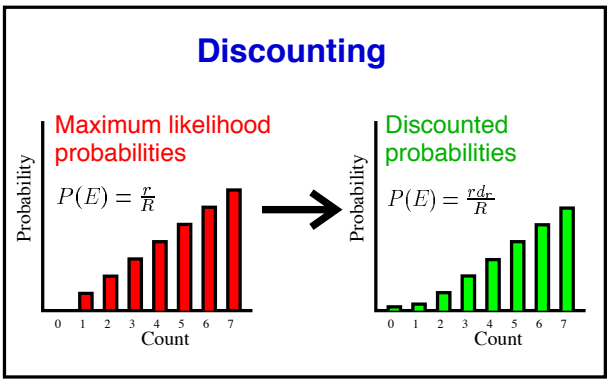
Two linked solutions discussed here:

- **discounting** allocating some "counts" to unseen events
- **backing-off** for rare events reduce the size of N

Discounting

- Need to reallocate some counts to unseen events
- **Must** be a valid PMF so for a trigram must satisfy

$$\sum_{k=1}^{V} \hat{P}(\omega_k|\omega_i,\omega_j) = 1$$



- General form of discounting (for a trigram)

$$\hat{P}(\omega_k|\omega_i,\omega_j) \quad = \quad d(f(\omega_i,\omega_j,\omega_k))\frac{f(\omega_i,\omega_j,\omega_k)}{f(\omega_i,\omega_j)}$$

- need to decide form of $d(f(\omega_i,\omega_j,\omega_k))$ (and ensure sum-to-one constraint)

Forms of Discounting

Notation: r=count for an event, n_r=number of N-grams with count r

Various forms of discounting (**modified Kneser-Ney** also popular)

- **Absolute** discounting: subtract constant from each count

$$d(r) = (r - b)/r$$

  Typically $b = n_1/(n_1 + 2n_2)$ - often applied to all counts

- **Linear** discounting:

$$d(r) = 1 - (n_1/T_c)$$

  where T_c is the total number of events - often applied to all counts.

- **Good-Turing** discounting: ("mass" observed once = n_1, observed r = rn_r)

$$r^* = (r + 1)n_{r+1}/n_r; \text{ probability estimates based on } r^*$$

  unobserved same "mass" as observed once.

Backing-Off

When N-grams cannot be estimated reliably because the count $f(w_i, w_j, \ldots)$ is too small,

- **back off** to a more general distribution
- normally based on an $(N-1)$-gram is used instead.

For the example of a bigram

$$\hat{P}(\omega_j|\omega_i) = \begin{cases} d(f(\omega_i,\omega_j))\frac{f(\omega_i,\omega_j)}{f(\omega_i)} & f(\omega_i,\omega_j) > C \\ \alpha(\omega_i)\hat{P}(\omega_j) & \text{otherwise} \end{cases}$$

$\alpha(w_i)$ is the **back-off weight**, it is chosen to ensure that

$$\sum_{j=1}^{V} \hat{P}(\omega_j|\omega_i) = 1$$

- The back-off weight is computed separately for each history and uses the $N-1$'th order N-gram count.

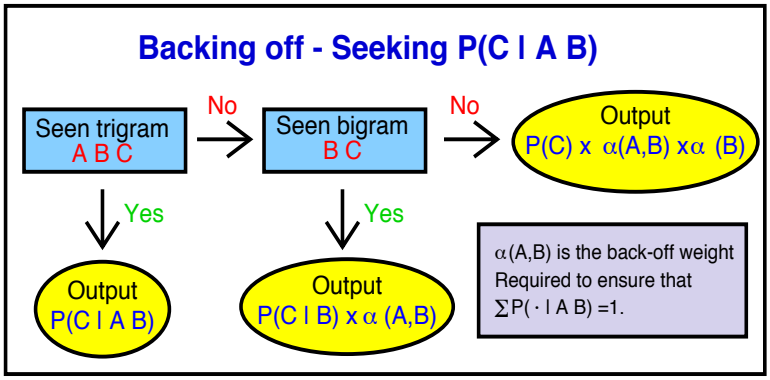
C is the N-gram **cut-off** point (can be set for each value of N)

- value of C also controls the size of the resulting language model

Backing-Off (cont)

Interested in extending beyond simple bigram to higher-order *N*-grams

- require a backoff-chain



- "Seen" means that the counts (trigram or bigram) exceed cut-off point, *C*
- this value may be set separately for the bigrams and trigrams.

Practical *N*-Gram Training

All language models that we will discuss are trained from a text corpus:

- desirable to have as large a corpus as possible:newspaper about 20MW (million words)/year; audio about 10k words/hour.
- corpus should be **representative** (as possible) of data that will be seen by recogniser.

Note that text needs to be "normalised" before use:

- Unusable parts of the data need to be discarded (e.g. tables)
- the data put into a standard format,
- sentences individually tagged
- numbers/dates processed appropriately in spoken form(s)

Practical *N*-gram modelling software needs to be able to process very large volumes of training data. A number of toolkits:

- process the data to gather sorted *N*-gram files.
- generate frequency-of-frequency statistics which are used to compute discount coefficients
- From these, the back-off weights and the final *N*-gram model is generated.

Some LM toolkits include

- SRILM: widely used, many discounting types, and LM variants
- KenLM: often used for machine translation LMs (but can also use for ASR!)
- HTK HLM toolkit (and simple HTK HLStats for very small corpora).

These toolkits allow LM generation from many 100's/1000's MW of text in a few hours.

Performance with varying *N*

Experiments on broadcast news (BN) data

- LM trained on about 230 MW of broadcast news transcriptions, newswire texts and acoustic transcriptions.
- Test on two BN data sets, BNdev96ue & BNeval97.

The recogniser uses an HTK state-clustered triphone GMM-HMM (maximum-likelihood trained) system (with MLLR adaptation) and a 65k word language model of various *N*.

Lang Model Type	Perplexity		% WER	
	BNdev96ue	BNeval97	BNdev96ue	BNeval97
bigram	243	240	27.9	21.3
trigram	172	159	24.6	18.0
4-gram	161	147	24.0	17.3

- Most gain is from bigram to trigram
- Different test sets have different word error rates independent of perplexity
- Here perplexity is a reasonable predictor of WER change (e.g. on BNeval97 reduction of 39% in perplexity leads 19% in WER)

N-Gram Variation with Cut-Offs

Data: training on 130MW of broadcast news data and testing on 18MW.

Cutoffs		Number of Parameters	Perplexity
Bigram	Trigram		
0	0	38,289,694	129.5
1	1	12,294,777	135.4
2	2	6,965,818	141.0
5	5	3,168,606	153.5
10	10	1,729,668	167.6
20	20	936,064	186.3
50	50	407,266	220.4
100	100	213,488	252.8

Depending on the homogeneity of the language model training/test data it is not unusual to find that using a cut-off of zero is **no-better** in terms of perplexity than a cut-off of 1 or 2.

Often the cut-off for higher order *N*-grams is higher than for lower order (e.g. cut-offs of 1,3,3 used for the 4-gram on previous slide).

An alternative to cut-offs is to use ***N*-Gram pruning** which removes higher order *N*-grams (& hence relies more on back-off) to minimise the increase in the training set entropy

Language Model Scaling

During recognition it is necessary to combine the language model probability with the acoustic model likelihood. The Bayes' formulation tells us that we should just multiply these quantities together (or add the logs).

However if this is done directly the effect of the language model will be relatively small because of the much higher dynamic range of the acoustic (log) likelihoods.

Therefore the language model log probabilities are scaled i.e. compute

$$\log p(\mathbf{O}|\mathbf{W}) + \alpha \log P(\mathbf{W})$$

where α is the **language model scale factor**. Note that scaling the log probability is equivalent to raising the log probability to a power. The same operation (if finding the best path) can be done by compressing the dynamic range of the acoustic model scores.

This mismatch in dynamic range has several sources including the (incorrect) independence assumptions in HMMs which (greatly) underestimate the acoustic likelihood.

You will find that this scaling procedure is rarely mentioned in the literature although everyone uses it!

Aside: for similar reasons, if pronunciation probabilities are used they also need to be scaled.

Mixture-Based Language Models

Often it is useful to use data from a variety of text sources and train a language model for each.

If these are used as component models, then a **mixture** of language models can be formed, whereby the language models are **interpolated** using component weights.

For a mixture of *N*-grams (but it could be a mixture including any type of LM that can give a word-by word probability) the overall word prediction probability is

$$P(w_k|h) = \sum_{n=1}^{M} \lambda_n P_n(w_k|h)$$

where h is the word history, M is the number of mixture components, and

$$\sum_{n=1}^{M} \lambda_n = 1, \quad \lambda_n \geq 0$$

Note that interpolation is performed at the probability level of a complete language model and is somewhat different to interpolating for smoothing an individual LM.

Multiple situations in which which mixture LMs may be useful include

- Explicit modelling of different **topics** in the data (one model for each topic)
- Modelling different **styles** of speech
- **Task specific** models

Note that the λ_n values could be adaptive.

Mixture models - Uses and Estimation

Important use when large quantities of task-specific LM training data not available

Use a **background** model that doesn't suffer from trained on more general text

e.g. LM for **spontaneous speech**

- amount of transcribed acoustic speech data is rather limited.
- background model(s) and task-specific model interpolated
- Often optimum weight depend on the *N*-gram order
- As *N*-gram order increases, increase weight on background model since sparseness is more of a problem.

Note that the mixture (interpolation) weights can be found automatically:

- An automatic approach as the number of components increases
- **Maximising the log likelihood** of a some development data by E-M.
- Since the number of interpolation weights estimated is normally small, only fairly small amount of data is needed for this purpose.

If using back-off models with linear interpolation then (a good approximation to) the resulting interpolated model can be **pre-computed** and a single combined word *N*-gram model results.

EM Procedure for LM Component Weights

Assume here for simplicity just two weights and trigram language models:

- need posterior probability of particular LM component generating the development data

Procedure for using E-M, with the weight estimated for models "a" (acoustic) and "n" (news), and assuming a trigram model

1. Initialise the set of weights (usually uniform distribution is used), $\tau = 0$
2. Given the current weights $\boldsymbol{\lambda}^{(\tau)}$, compute probability of each component for each word k in the development data:

$$P(a|\mathbf{W}, k, \tau) = \frac{\lambda_a^{(\tau)} P_a(w_k|w_{k-2}, w_{k-1})}{\lambda_a^{(\tau)} P_a(w_k|w_{k-2}, w_{k-1}) + \lambda_n^{(\tau)} P_n(w_k|w_{k-2}, w_{k-1})}$$

3. Using EM the $\tau + 1^{th}$ estimate of the interpolation weight for model a is then given by

$$\lambda_a^{(\tau+1)} = \frac{1}{K+1} \sum_{k=1}^{K+1} P(a|\mathbf{W}, k, \tau)$$

4. $\tau = \tau + 1$, goto (2) unless converged or maximum iterations reached

Note that this can easily be extended to multiple (> 2) component models

Class-Based Models

Can also reduce sparseness by reducing number of history **equivalence classes** by grouping words into classes.

If each word belongs to a single class, for a **class-bigram** model, probability $P(\omega_j|\omega_i)$ is

$$P(\omega_j|\omega_i) = P(c_j|c_i)P(\omega_j|c_j)$$

where c_j is the class containing ω_j.

- Word histories with same sequence of classes are viewed as equivalent
- $P(\omega_j|c_j)$ gives the probability of of the word given its class (total number of these **class-conditioned unigrams** is V over the whole vocabulary).
- If each word belongs to its own class, then the model above becomes a word model.

If relatively **few classes** used, then a longer N-gram context may be used.

Normally

- Words are members of a single class only (otherwise much more complex to use)
- Class membership by automatic clustering algorithms based on bigram statistics
- A few hundred classes to one thousand word classes might be used
- Can interpolate with a word model

Summary

Language Models are a key component in large vocabulary speech recognition systemsa

- Reduce the equivalent average number of word choices by several orders of magnitude
- Normally simple models based on ***N*-grams** are used
- Base prediction on previous N-1 words. Typically $N = 1 \ldots 4$.
- Need to deal with data **sparseness**: most N-grams don't occur in training data
- Use a combination of **discounting** and **backoff**
  - Various discounting methods
  - Can control model size by cut-offs or entropy based pruning
- Mixture models interpolate several language models: can be used to
  - Provide task-specific models (e.g. spontaneous speech)
  - Model topics
  - Combine class-based and word-based N-grams
- Class-based models
  - Far fewer history equivalence classes
  - Normally use automatic clustering
  - Interpolated word and class N-grams can provide reductions in WER

These ideas will be further extended when dealing with **neural network LMs**