Statistical Language Modelling

Fethi BOUGARES

fethi.bougares@univ-lemans.fr
Laboratoire d'Informatique de l'Université du Maine

some slides from Mohit Iyyer

Language Model - What and why?

Aims of language models

- Predict the future! ... words
- Assign a probability to a sentence (or sequence of words)

Many applications

- Speech recognition
  - Lame aise on bleu
  - Là mais omble eux
  - La mais on bleue
  - La maison bleue $\leftarrow$ This one is more probable!
  - La maison bleu
- Machine Translation: "The blue house"
  - "La bleue maison" $\leftarrow$ feels like not natural (less probable)
  - "La maison bleue" $\leftarrow$ this one seems better! (more probable)

Language Model - LM

- Allows to distinguish between well written sentences and bad ones
- Should give priority to grammatically and semantically correct sentences
  - in a implicit fashion, no need for a syntactic nor semantic analysis
  - Monolingual process $\rightarrow$ no need of annotated data

Language Model - LM

- Goal: provide a non zero probability to **all** sequences of words
  - even for non-grammatical sentences
  - learned automatically from texts

Issues:

- How to assign a probability to a sequence of words?
- How to deal with unseen words and sequences ?
- How to ensure good probability estimates?

Language Model

- Goal: provide a non zero probability to all sequences of words W extracted from a **vocabulary** $|V|$
- Vocabulary: list of all words known by the model
  - a specific word **<unk>** to manage all the words not in V
  - word $=$ sequence of characters without space
  - word $\neq$ linguistic word $\rightarrow$ token

Let $W = (w_1, w_2, \ldots, w_n)$ with $w_i \in V$ be a word sequence

LM - Complexity

- Complexity for a vocabulary size of 65k
  - $65k^2 = 4\ 225\ 000\ 000$ sequences of 2 words
  - $65k^3 = 2.74 \times 10^{14}$ sequences of 3 words

  $\rightarrow$ French: 3k words in the fundamental dictionary, 30k for general knowledge

$\rightarrow$ We can't directly estimate the probability of a sequence by relative frequency!

- Equivalence classes
  - group histories in equivalence classes ϕ

$$p(W) \approx \prod_{i=1}^{T} p(w_i|\phi(h_i))$$

  - Language modelling lies in determining ϕ and find a method to estimate the corresponding probabilities

LM - n-gram

- n-gram: sequence of n words
    - Ex.: "La maison bleue est jolie"

  $\rightarrow$ bi-grams: "La maison", "maison bleue", "bleue est", "est jolie"

  $\rightarrow$ tri-grams: "La maison bleue", "maison bleue est", "bleue est jolie"

  $\rightarrow$ 4-grams: "La maison bleue est", "maison bleue est jolie"

- n-gram model uses an equivalence class mapping the history h_i to the $n-1$ previous words

LM - Probabilities

- How to estimate the n-gram probabilities?
- Maximum Likelihood Estimation (MLE)
  - Get counts from a **corpus**
  - **normalize** them so that they are between 0 and 1
- Unigram probabilities

$$p(w_i) = \frac{C(w_i)}{\sum_k C(w_k)} = \frac{C(w_i)}{\text{corpus size}}$$

$\rightarrow$ $C(.)$ is the counting function

- n-gram probabilities

$$p(w_i|h_i^n) = \frac{C(h_i^n w_i)}{C(h_i^n)}$$

LM - Probabilities / Example

Corpus

- <s> une maison bleue </s>
- <s> une maison grise </s>
- <s> la table grise est dans la maison bleue </s>

- Probabilities of some bi-grams:
  - $P(une|\text{<s>}) = \frac{2}{3} = 0.67$; $P(la|\text{<s>}) = \frac{1}{3} = 0.33$; $P(\text{</s>}|bleue) = \frac{2}{3} = 0.67$
  - $P(grise|maison) = \frac{1}{3} = 0.33$; $P(bleue|maison) = \frac{2}{3} = 0.67$
- Probabilities of some tri-grams:
  - $P(maison|\text{<s>}\ une) = \frac{2}{2} = 1$; $P(bleue|une\ maison) = \frac{1}{2} = 0.5$

LM - n-gram

- bigram model: $\phi(h_i) = (w_{i-1})$

$$p(W) \approx p(w_1) \times \prod_{i=2}^{T} p(w_i|w_{i-1})$$

- trigram model: $\phi(h_i) = (w_{i-1}, w_{i-2})$

$$p(W) \approx p(w_1) \times p(w_2|w_1) \times \prod_{i=3}^{T} p(w_i|w_{i-1}, w_{i-2})$$

- n-gram: $\phi(h_i) = (w_{i-1}, \ldots, w_{i-n+1})$
- Consequences:
  - $n-1$ words are enough to predict the next word $\leftarrow$ **Markov** assumption.

LM - Sequence probability

- How to compute the probability of a sequence ?

→ By combining the n-gram probabilities!

$$p(W) = \prod_{i=1}^{T} p(w_i|\phi(h_i))$$

with $h_i = (w_1, w_2, \ldots, w_{i-1})$ the history of word w_i

with $\phi(.)$ the function mapping the history to the equivalence classes of size $n-1$

- in practice: n ranges to 4 or 5, barely 6 ⇒ require exponential quantity of data

Example: bi-gram P(<s> la maison grise </s>)

- P(.) = P(la|<s>) * P(maison|la) * P(grise|maison) * P(</s>|grise)
  = 0.33 * 0.5 * 0.33 * 0.33 = 0.0179685

LM - Characteristics

- Language structure implicitly captured
  - probability of succeeding words, cooccurrences
  - same for semantic
- Probabilities are independent from the position in the sentence
  - add begin (<s>) and end (</s>) of sentence tokens
- Probabilities are estimated using a large quantity of data (corpus), which are supposed to be **well written**

LM - Unseen sequences

- Wrong sequences that are not allowed by the language
  - Ex.: "ils part tôt", "elle est beau"
- Correct sequences that are not seen in the training corpus

→ How to avoid a zero probability?

Solutions

- Increase training corpus size

→ makes training longer + can we ever get a perfect corpus?

- Reserve a (small) probability mass to unseen events

$$\epsilon \geq p(w_i | h_i^n) > 0 \ \forall i, \forall h$$

→ This is **smoothing** or **discounting**

LM - Smoothing

- Idea: take probability mass D to seen events, then redistribute it to unseen events

$\rightarrow$ smoothing

- Laplace smoothing (also known as **add 1** smoothing)

$$P_{Laplace}(w_i) = \frac{C(w_i) + 1}{corpus\ size + V}$$

- **add-k** smoothing:

$$P_{add-k}(w_i) = \frac{C(w_i) + k}{corpus\ size + kV} \text{ with } 0 < k < 1$$

LM - Smoothing

- Kneser-Ney smoothing: absolute discounting and continuation
- absolute discounting: subtract a certain (fixed) quantity to the counts
- continuation: words seen in more contexts are more likely to appear in a new context
    - Ex.: In the corpus "Mans" is more frequent than "table"
    - but seen only in the context of "Le Mans", while "table" has many more contexts

  → so higher probability of continuation

$$P_{kn}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) - d}{C(w_{i-1})} + \lambda(w_{i-1})P_{cont}(w_i)$$

- Going further: comparative study

→ Stanley F. Chen and Joshua T. Goodman, *An Empirical Study of Smoothing Techniques for Language Modelling*. Computer, Speech and Language, 13(4), pp. 359-394, 1999.

LM - Backoff

- Idea: exploit lower order history
- Backoff technique

$$\tilde{p}(w_i|h_i^n) = \begin{cases} p(w_i|h_i^n) & \text{if } C(h_i^n w_i) > 0 \\ \alpha(h_i^n)p(w_i|h_i^{n-1}) & \text{if } C(h_i^n w_i) = 0 \end{cases}$$

- with $\alpha(h_i^n)$ the backoff weight

$\rightarrow$ computed so that probability distribution is respected (probs between 0 and 1 and sums to 1)

- See [Jurafsky and Martin, 2018]

LM - In practice

- How to set the vocabulary?
- Machine Translation:
  - Use all the ~~words~~ tokens belonging to **in domain** corpora

  → target side of train and development corpora

  → specialized monolingual corpora
  - Most frequent ~~words~~ tokens of large generic corpora

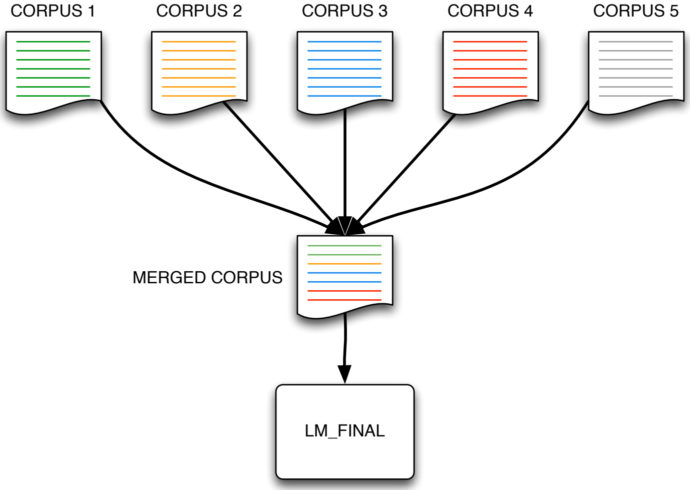
  → seen at least k times

- Speech recognition:
  - Only consider words than the speech decoder can produce

  → map all others to <unk>

LM - Training methodology

- Merge training data, standard training procedure

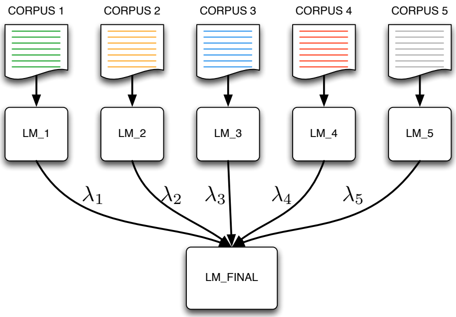


LM - Training methodology

- (log) linear interpolation
- with J models:

$$p(w_i|h_i^n) = \sum_{j=0}^{J} \lambda_j \cdot p_j(w_i|h_i^n)$$

$\rightarrow$ λ_j are computed using an EM procedure



LM - perplexity

- Perplexity: measures the ability of the model to predict word of an unseen text
  - unseen = not in training data
  - criterion to be minimized
  - Let's consider a corpus $W = w_1 w_2 ... w_N$

$$
\begin{aligned}
PPL(T) &= P(w_1 w_2 ... w_N)^{-\frac{1}{N}} \\
&= \sqrt[N]{\frac{1}{P(w_1 w_2 ... w_N)}} \\
&= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 ... w_{i-1})}}
\end{aligned}
$$

LM - perplexity and cross entropy

$$
\begin{aligned}
PPL(T) &= 2^{H(W)} \qquad [H \text{ is the cross-entropy}] \\
H(T) &= -\frac{1}{N} \log P(w_i | w_1 ... w_{i-1})
\end{aligned}
$$

- Perplexity is linked to the number of possible next words that can follow any word

$\rightarrow$ How accurately can the model predict the next word?

Example with the digits (zero, one, two, ..., nine)

- if equilibrated corpus, i.e. $P = \frac{1}{10}$ for all words

$$PPL(W) = P(w_1 w_2 ... w_N)^{-\frac{1}{N}} = \left(\frac{1}{10^N}\right)^{-\frac{1}{N}} = \left(\frac{1}{10}\right)^{-1} = 10$$

- This quantity would reduce if a word is more frequent

$\rightarrow$ ex: "zero" is 10 times more frequent

Dealing with OOVs and low frequency words

- Replace all words **not** in the vocabulary by the token <unk>
- Compute <unk> probability similarly to the other words.
- Apply this rule to words appearing less than n times in the training corpus.

$\rightarrow$ Allows to select and fix the vocabulary size

- Choice of words mapped to <unk> impacts perplexity

$\rightarrow$ a small vocabulary and a larger <unk> probability will reduce the perplexity

$\Rightarrow$ Always compare perplexities of models having same vocabulary!

Why do n-grams work so well?

- Probabilities computed on a large corpus

$\rightarrow$ the more the better

- Implicit modelling of syntax and semantics

$\rightarrow$ Correct word sequence are more probable!

$\rightarrow$ e.g. number and gender agreement

- Easy to integrate into search methods like Viterbi

Issues with n-grams models

- Cannot model long distance dependencies
  $\rightarrow$context size is limited (up to 6 token in practice)

- Poor modelling of:
  - new vocabulary words
  - domain
  - unprepared text (discourse)

$\rightarrow$ Do not capture the meaning

Count-based Language Models Cannot handle long-distance dependencies!!

Example :

for programming class he wanted to buy his own computer
for tennis class he wanted to buy his own racquet

Beyond Count-based Language Models

Neural Language Modelling

How to build a neural Language Model?
Recall the Language Modeling task:

1. **Input:** sequence of words : $x^{(1)}, x^{(2)}, ..., x^{(n)}$
2. **Output:** probability distribution of the next word $P(x^{n+1} = w_j | x^(1), x^{(2)}, ..., x^{(n)})$

How about a window-based neural model?

Neural Language Modelling

~~as the proctor started the clock~~ $\underbrace{\textit{the students opened their}}_{\text{fixed window}}$ ______

discard

Fixed-window neural Language Model

output distribution:

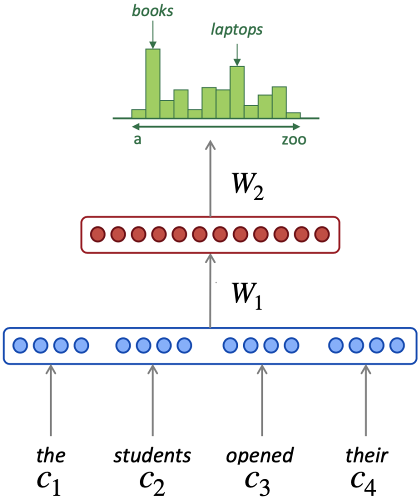
$$\hat{y} = softmax(W_2 h + b_2)$$

hidden layer

$$h = f(W_1 c + b_1)$$

concatenated word embeddings

$$c = [c1; c2; c3; c4]$$



Fixed-window neural Language Model

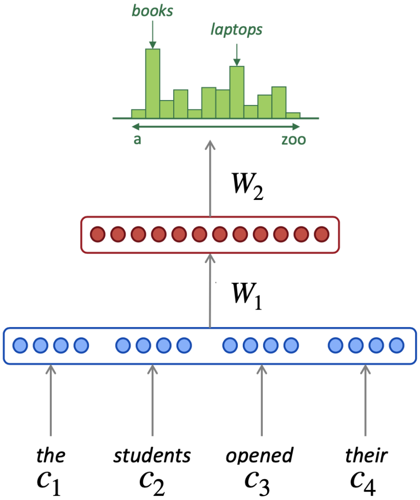
Compared to N-gram LM

Improvements :

- No sparsity problem
- Model size is $O(n)$ not $O(exp(n))$

Remaining problems: :

- Fixed window is too small
- Enlarging window enlarges W
- Window can never be large enough!
- Each c_i use different rows of W. (We don't share weights across the window)

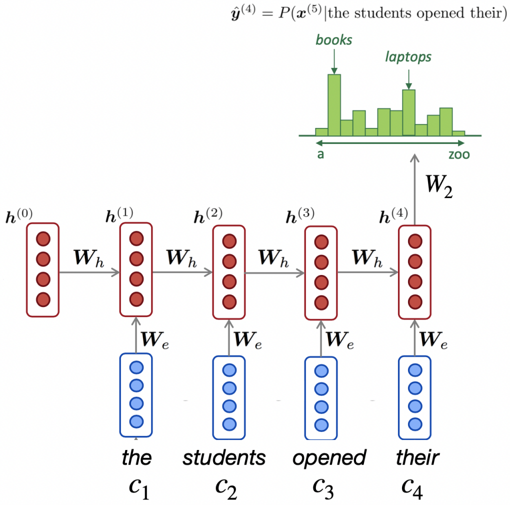


What about Recurrent Neural Networks!

Recurrent neural Language Model

output distribution:
$\hat{y} = softmax(W_2 h^{(t)} + b_2)$
hidden states :
$h^{(t)} = f(W_h h^{(t-1)} + W_e c_t + b_1)$

$h(0)$ is the initial hidden state
word embeddings :
$c_1; c_2; c_3; c_4$



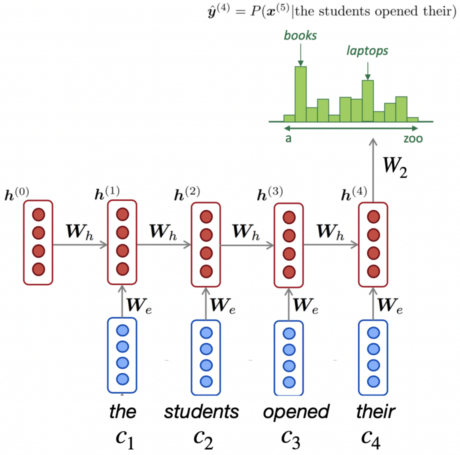
Recurrent neural Language Model

RNN advantages

- Can process any input length
- Model size doesn't increase for longer input
- Weights are shared across timesteps

RNN Disadvantages

- Recurrent computation is slow
- In practice, difficult to access information from many steps back
- Weights are shared across timesteps



Training a RNN Language Model

- Get a big corpus of text which is a sequence of words
- Feed into RNN-LM; compute output distribution $\hat{y}^{(t)}$ for every step t.
- Loss function on step t is usual cross-entropy between our predicted probability distribution $\hat{y}^{(t)}$ and the true next word $y^{(t)} = x^{(t+1)}$

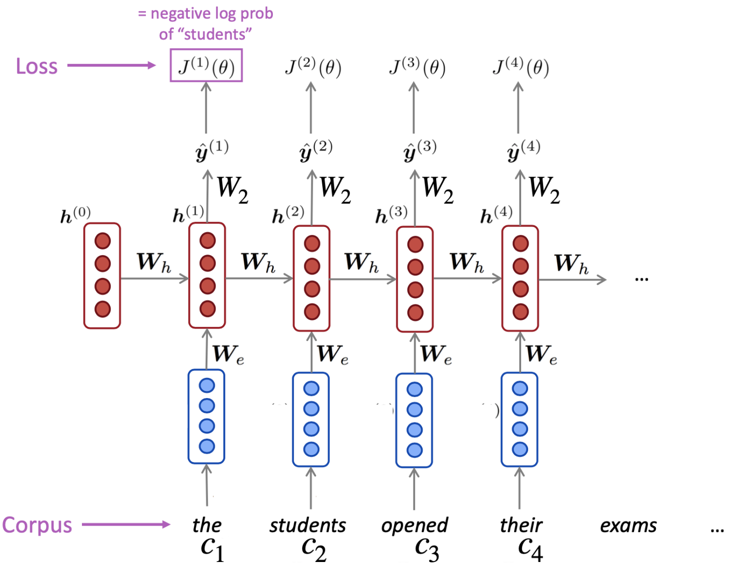
$$J^{(t)}(\theta) = CE(y^{(t)}, \hat{y}^{(t)}) = -\sum_{j=1}^{|V|} y_j^{(t)} log\ \hat{y_j}^{(t)}$$

  $\Rightarrow$ predict probability dist of every word, given words so far

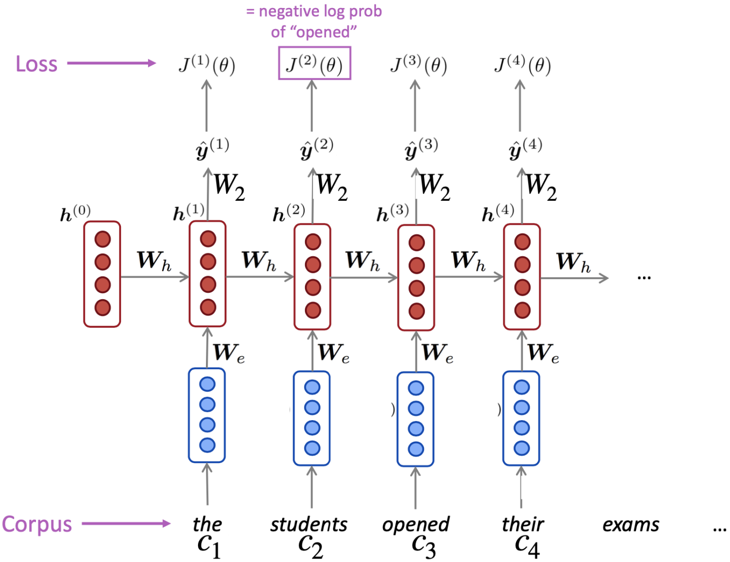
- Average this to get overall loss for a batch of T examples):

$$J(\theta) = -\frac{1}{T}\sum_{j=1}^{|T|} J^{(t)}(\theta)$$

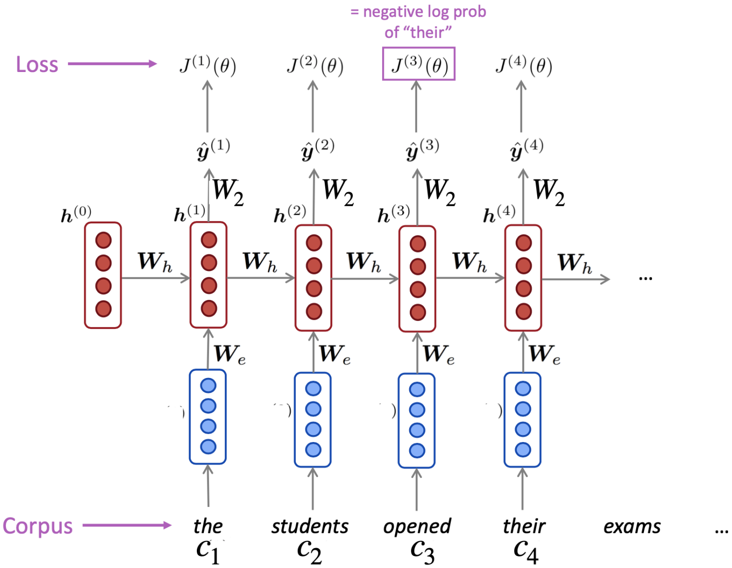
Neural Language Modelling



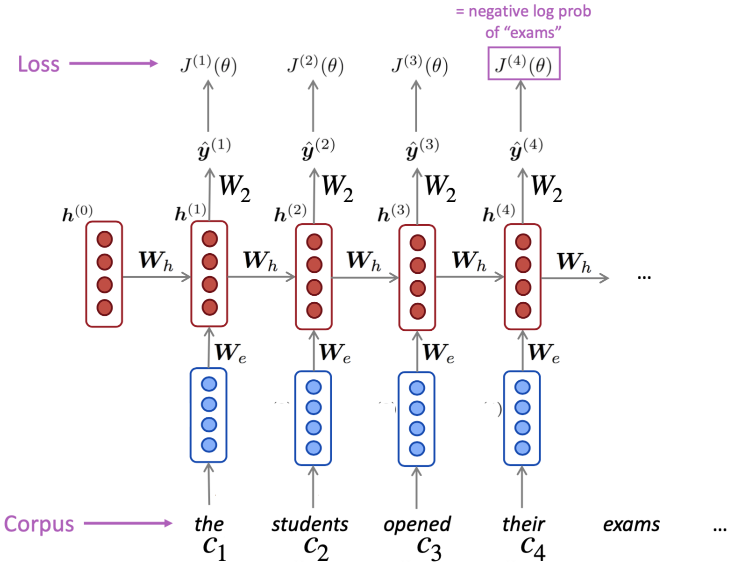
Neural Language Modelling



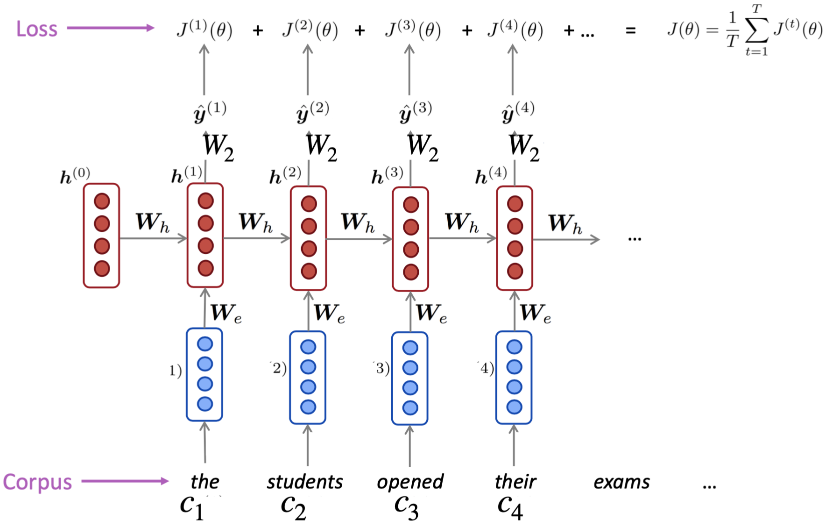
Neural Language Modelling



Neural Language Modelling



Neural Language Modelling



LM - toolkit - SRILM

- See [Stolcke, 2002]
- Build a model: ngram-count

→ !! always specify order with **-order N** and use of unknown class with **-unk**

- Compute interpolation weights: compute-best-mix

  use the outputs of the following command: **ngram -debug 2 -ppl ...**
- Compute perplexity, interpolated model: ngram

  -ppl <dev corpus>: compute perplexity on development corpus

  -mix-lmK <lmK> -mix-lambdaK <coeffK> : interpolate several models <lmK> with weights <coeffK> (K ranging from 0 to 9)

References I

Jurafsky, D. and Martin, J. H. (2018).
Speech and language processing.
In *https://web.stanford.edu/%7Ejurafsky/slp3/ed3book.pdf*.

Shannon, C. E. and Weaver, W. (1948).
A Mathematical Theory of Communication.
University of Illinois Press, Champaign, IL, USA.

Stolcke, A. (2002).
Srilm - an extensible language modeling toolkit.
Proceedings of the 7th International Conference on Spoken Language Processing (ICSLP 2002), pages 901–904.