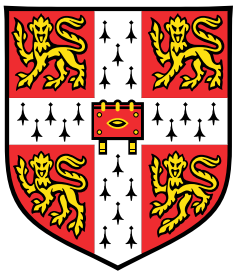
Combining Diverse Neural Network Language Models for Speech Recognition



Xianrui Zheng

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy in Machine Learning and Machine Intelligence

Queens' College August 2019

I would like to dedicate this thesis to my loving parents ...

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this thesis are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 15,000 words including appendices, bibliography, footnotes, tables and has fewer than 150 figures.

This project uses some resources from others. Guangzhi Sun provided the forward RNNLM script, Qiujia Li provided an example of how CMA-ES strategy can be used and Chao Zhang provided the 50-best list generated by an ASR system. All provided scripts are extended so that they fit in my framework. More explanation and details about the scripts created from scratch, extensions made in source codes provided by others and packages used are included in Section 5.2.

Xianrui Zheng
August 2019

Acknowledgements

I would like to thank Prof Woodland and Dr Chao Zhang for their constant support throughout the project. This project allows me to explore how fascinating the language modelling is and its wide applications in nature language processing field. Not only did the supervisors teach me the knowledge related to this thesis, but they also helped me to develop good behaviours for doing research. I also want to thank Guangzhi Sun, Qiujia Li and Florian Kreyssig for their kind support.

Abstract

Language modelling is a crucial part of speech recognition systems. It provides the prior information of sequences to the automatic speech recognition (ASR) system, which allows maximum a posteriori decoding. Count based n-gram language models (LM) with various smoothing techniques have been popular for many years.

Due to the development of hardware and software, the implementation of neural networks has become increasingly efficient, so that researchers have started to use neural networks to model texts. Neural network language models (NNLMs) have proven to be better at modelling the meanings and relationships between words or sub-words. They can also mitigate the data sparsity issue in n-gram models and take much longer history into consideration. Since different NNLMs have different assumptions and different architectures, they may be better at modelling different aspects of language, which means they can be highly complementary. Therefore, this project focuses on different methods to combine NNLMs. All models are combined via interpolation and different ways to find the weights for each model are investigated.

Pre-trained NNLMs have become very popular in recent years. They are trained on much larger corpora and can be fine-tuned by using in-domain data with specific tasks. Previous work has shown that fine-tuning pre-trained models can achieve state-of-the-art results in many NLP tasks such as machine translation and question answering. However, no prior work using such pre-trained models to perform n-best rescoring task for speech recognition was found. This thesis will investigate the performance of some pre-trained models before and after combining with NNLMs that were trained by using in-domain data only (which are referred to as normal NNLMs).

The experiments in this project show that after the combination of normal NNLMs, the word-error-rate (WER) is lower than that from any single model. By testing on AMI corpus, an absolute decrease of 0.5% in WER is observed with eval set by comparing the combined model with the best NNLM without combination. After interpolating normal NNLMs with fine-tuned pre-trained models, the best result in this thesis was achieved with a further 0.9% absolute reduction in eval WER. The combination techniques used in this thesis can be easily extended to other corpora, or to involve more NNLMs for combination.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Speech sits at the centre of communication in our society. Modern technologies for analysing speech require knowledge from fields including but not limited to linguistics, engineering and psychology. Significant progress has been made in the past.

One of the most important areas of speech research is speech recognition, which has been attracting interest over a long period. Modern Automatic Speech Recognition (ASR) systems are able to transcribe human speech into text with a large vocabulary ($\sim$100K words). The ASR system has a wide range of applications, such as broadcast news transcription (Graff et al., 1997; Woodland et al., 1997a,b) and telephone conversation transcription (Saon et al., 2016). Generating transcriptions or taking notes for meetings is another demanding application of ASR (Yu et al., 2000). In most cases, ASR systems need to handle the speech from speakers with different speaking styles and from different environments with varying levels of background noise.

Various models have been applied in speech recognition tasks. Due to the improvement of hardware, more data can be processed and more computationally intensive neural network models can be applied to this area (Hinton et al., 2012).

With only acoustic information, a speech recognition system may not be able to correctly identify words, since many words can have similar pronunciations. To improve system accuracy, we want the system to consider neighbouring words. Therefore, for any input speech, an ASR system combines the probability of each word provided by an acoustic model and the probability of the word sequence determined by a language model. The acoustic model can be an HMM-based model (Gales and Young, 2007; Rabiner, 1989). It usually uses phone-based models which gives the probability that a given segment of speech is generated by a particular phone. The language model can be a n-gram model that is trained with a large text corpus. This thesis focuses on the language modelling part of speech recognition. In

particular, various neural network language models are explored and combined together for n-best LM rescoring for speech recognition.

Word Error Rate (WER) is a common way to measure the performance of an ASR system. Recent advances in language modelling have reduced the WER significantly (Kombrink et al., 2011; Mikolov and Zweig, 2012; Wen et al., 2013). However, there is still room for improvement. NNLMs are highly complementary because they all have their own assumptions embedded into model architectures. Therefore, combining them helps to gather the advantages of each single model and can improve the rescoring performance. The n-gram model suffers from data sparsity issue but the parameters can be efficiently estimated with a large amount of data. A longer history is often helpful for determining the probability of a word sequence. Some state-of-the-art neural network language models (NNLM), such as the recurrent neural network language model (RNNLM) and the Transformer language model (TLM), have the ability to consider much longer histories. The combination of these models helps to combine their strengths and leave their weaknesses behind.

Many pre-trained language models for general purpose use have been recently released (Dai et al., 2019; Devlin et al., 2018; Peters et al., 2018; Radford et al., 2018, 2019; Yang et al., 2019). Due to the increase of computational power and the growth of text datasets available on the Internet, they can be trained on large datasets with billions of words. After fine-tuning with in-domain datasets, they are able to be optimized for a specific task. Many promising results have been discovered in fields such as machine translation and reading comprehension. Such models are also fine-tuned in this project and combined together with other NNLMs.

1.1 Thesis Outline

The structure of the thesis is as follows:

- Chapter 2 reviews ASR systems and how language models are used.
- Chapter 3 introduces different types of neural network language models in detail, including feed-forward NNLMs, Recurrent NNLMs and Transformer NNLMs.
- Chapter 4 outlines 3 state-of-the art pre-trained models with different methods for pre-training.
- Chapter 5 presents the implementation methods and experimental results.
- Chapter 6 concludes this thesis and discusses the possible future work.

1.2 Contribution

This thesis endeavours to combine more neural network language models than previous work, where typically only one NNLM was used or a combination of one NNLM and a n-gram model was used for speech recognition language model rescoring task. Also, pre-trained models are used as part of the combination. The best combined model achieves much better results than any single model tested in this project, which shows the potential of combining diverse NNLMs for speech recognition.

Chapter 2

Automatic Speech Recognition (ASR)

This chapter first reviews two components in a traditional HMM-based ASR system, namely the acoustic model and the language model, followed by how they are used together.

2.1 Acoustic Modelling with Hidden Markov Model

Hidden Markov Model (HMM) based acoustic models have been widely used for many years. The raw waveform is first converted to acoustic feature vectors. Each vector represents a segment of speech lasting around 25 milliseconds. Mel-filterbank with Discrete Cosine Transformation, which is called Mel-Frequency Cepstral Coefficients (MFCC) (Davis and Mermelstein, 1980), is a popular method for extracting the vectors. The first and second order differentials of the static MFCC vectors can be used together for augmentation. Other methods such as perceptual linear predictive (PLP) (Hermansky, 1990) are also widely used.

Each phone has an associated HMM model, which normally has around 5 states. The first and the last states are non-emitting states and other states in the middle are emitting states. The existence of the non-emitting states allows HMM phone models to join together. Figure 2.1 shows an example of the HMM phone model. Here a_{ij} is the transition probability from state i to state j. Each state has an output probability defined by $b_j(o_t)$. The joint probability of the acoustic vector sequence O and the state sequence X given an acoustic model M in Figure 2.1 can be computed as

$$P(O,X|M) = a_{12}b_2(o_1)a_{22}b_2(o_2)a_{23}b_3(o_3)\ldots a_{45}b_5(o_6) \tag{2.1}$$

In general, with state sequence $X = \{x(0), \ldots, x(T)\}$, the joint probability is:

$$P(O,X|M) = a_{x(0)x(1)} \prod_{t=1}^{T} b_{x(t)}(o_t) a_{x(t)x_{t+1}} \tag{2.2}$$

The Viterbi algorithm is one option to efficiently find the unknown sequence of states which maximises the probability $P(O,X|M)$.

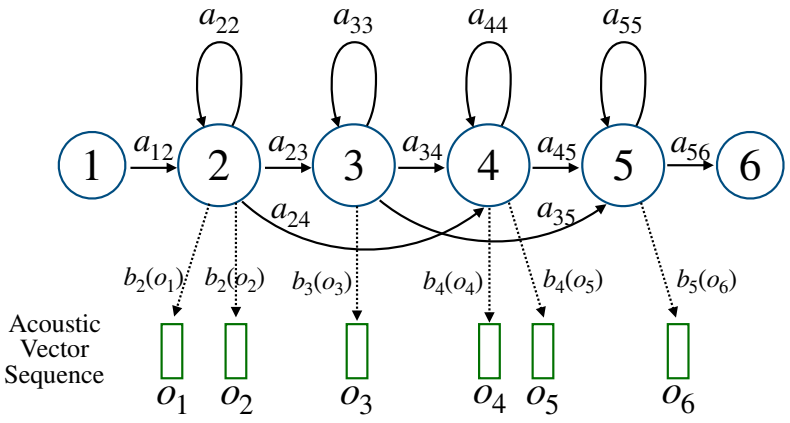


Fig. 2.1 A six states HMM acoustic model for a particular phone (Young et al., 2015)

The output probabilities given any state can be found by using either Gaussian Mixture Models (GMMs) or a Deep Neural Networks (DNNs). With a DNN, the output values of the neural network is the state posterior probabilities, which can be converted to the observation likelihood using Bayes rule. In particular, the DNN can be the time delay neural neural network (TDNN) (Waibel et al., 1995). A context window is used to collect information from multiple frames to construct inputs for TDNN. Each succeeding kernel also uses a context window to collect outputs from different time-steps of the current kernel. The kernel here can be one fully connected layer. Therefore, TDNN starts from looking at narrow temporal contexts and increase the size of temporal contexts in later layers. Originally the context window shifts one frame in time, but it is later shown that the window can shift more frames (Peddinti et al., 2015). Kreyssig et al. (2018) extends TDNN by adding more fully-connected layers and residual connections. Each kernel changes from one fully connected layer to three fully connected layers, and the last two fully connected layers in each kernel can be bypassed by a residual connection.

2.1.1 Context-Dependent Acoustic Modelling

An isolated speech sound may be influenced by its neighbouring speech sounds. The acoustic model can be extended to take care of the contextual effects such as coarticulation. For example, a tri-phone model can be used, which can model the content-dependent units. Instead of considering only the phone a, the tri-phone model takes the preceding phone and the succeeding phone of into consideration.

However, there are many possible tri-phones, which can lead to the data sparsity problem. State tying can be a solution (Bahl et al., 1991; Young et al., 1994), which ties states that are acoustically indistinguishable. Using phone decision trees (Young and Woodland, 1994) is one technique to determine which states to tie. The idea is to use a binary tree for each phone and state position. Each node in the tree asks a phonetic question from a large set of possible contextual effects, such as "is the right context consonant?", and is selected from a pre-defined question set by maximising the likelihood increase by splitting the node into two. Using such questions incorporates phonetic knowledge effectively into a data driven based clustering method, and allows the unseen tri-phones to be tied based purely on the selected questions once the trees are constructed.

2.1.2 Training Criterion

Minimum phone error (MPE) criterion (Povey and Woodland, 2002) is used for the discriminative training of HMM acoustic model in this project. Suppose there are R training observation sequences $\{O_1, \ldots, O_R\}$, the objective to be maximised is:

$$\mathcal{F}_{\text{MPE}}(\lambda) = \sum_{r=1}^{R} \frac{\sum_s P_\lambda(O_r|s)^\kappa P(s)^\kappa \text{RawAccuracy}(s)}{\sum_{s'} P_\lambda(O_r|s')^\kappa P(s')^\kappa} \tag{2.3}$$

where λ is the parameters in HMM, $\frac{P_\lambda(O_r|s)^\kappa P(s)^\kappa}{\sum_{s'} P_\lambda(O_r|s')^\kappa P(s')^\kappa}$ is the posterior sentence probability scaled by κ, which measures the probability of sentence s being the correct one. $P(s)$ is the probability from language model.

2.2 N-gram Language Models

Given a sequence of words $\{w_1, w_2, w_3, \ldots, w_t\}$ (abbreviated as $w_{1:t}$), n-gram language models predict the i^{th} word by using the previous $n-1$ words.

In particular, a tri-gram model predicts the current word (w_n) based on the previous two words ($w_{n-2:n-1}$). For a count-based language model, a simple way to estimate the

probability is to use maximum likelihood, which finds the relative frequencies:

$$P(w_n|w_{n-2:n-1}) = \frac{C(w_{n-2:n})}{C(w_{n-2:n-1})} \tag{2.4}$$

where C$(\cdot)$ finds the frequency of the sequence in the training dataset.

However, data sparsity is a serious issue in language modelling. If a vocabulary has 10000 words, then there would be $10000^3 = 1$ billion possible tri-grams. Therefore, it is not possible to have examples of many events. The event can be a sequence of three words. If an event in a test set is unseen in the training set, Equation 2.4 would simply assign a zero probability to that event. The speech recognition system would have no chance to output the correct hypothesis if a word sequence contains these unseen events.

Smoothing methods can mitigate the data sparsity problem. The relative frequencies obtained from seen events can be subtracted and given to unseen events to prevent zero probability during test time. Instead of assigning a zero probability to an unseen n-gram, the n-grams with lower orders can be used to estimate the probability. A simple way is to linearly interpolate individual models trained with different n-gram order. In the case of tri-gram, the final conditional probability given history can be computed as:

$$\tilde{P}(w_i|w_{i-1:i-2}) = \lambda_1 P(w_i|w_{i-1:i-2}) + \lambda_2 P(w_i|w_{i-1}) + \lambda_3 P(w_i) \tag{2.5}$$

where $P(w_i|w_{i-1:i-2})$, $P(w_i|w_{i-1})$ and $P(w_i)$ are estimated using maximum likelihood, and λ_i can be estimated by maximising the likelihood of a held-out set. Since $\tilde{P}(w_i|w_{i-1:i-2})$ needs to be a valid probability, the constrain for Equation 2.5 is $\sum_i^3 \lambda_i = 1$.

Back-off methods are also smoothing techniques that can estimate probabilities from sparse data. These methods consider the lower order n-gram models when an n-gram is unseen in the training set, which is different from interpolated models since back-off models do not use lower order n-gram models when an n-gram has positive counts. Two well-known back-off methods are reviewed:

2.2.1 Katz Smoothing

Katz (1987) extends the Good-Turing discounting method (Good, 1953). Let n_r be the number of n-grams that appear r times, i.e.

$$n_r := \sum_{w_{t-n+1:t}:\, c(w_{t-n+1:t})=r} 1$$

where $c(w_{t-n+1:t})$ is the number of times the n-gram $w_{t-n+1:t}$ appears in the training set. When the number of words in the dataset is N, Turing's estimate for the probability of a particular n-gram $w_{t-n+1:t}$ is

$$P_T = \frac{c^*(w_{t-n+1:t})}{N} \tag{2.6}$$

where

$$c^*(w_{t-n+1:t}) = (c(w_{t-n+1:t}+1))\frac{n_{c(w_{t-n+1:t})+1}}{n_{c(w_{t-n+1:t})}} \tag{2.7}$$

The ratio between $c^*(w_{t-n+1:t})$ and $c(w_{t-n+1:t})$ is denoted as $\frac{c^*(w_{t-n+1:t})}{c(w_{t-n+1:t})}$. Instead of directly using this ratio as the discount coefficient for an n-gram that appears in the training set. Katz (1987) suggests a modified discount coefficient d_r, such that

$$d_r = 1, \quad \text{for } r > k \tag{2.8}$$

which means no discount will be applied to counts greater than k and $k = 5$ is a good choice in practice (Katz, 1987).

For $1 \leq r \leq k$, The equation

$$\sum_{1 \leq r \leq k} n_r(1-d_r)\frac{r}{N} = \frac{n_1}{N} \tag{2.9}$$

needs to be satisfied. Applying this constrain to the equation

$$(1-d_r) = \mu\left(1-\frac{r^*}{r}\right) \tag{2.10}$$

where $r^* = (r+1)\frac{n_{r+1}}{n_r}$, the unique solution to d_r is

$$d_r = \frac{\frac{r^*}{r} - \frac{n_{k+1}}{n_1}}{1-\frac{(k+1)n_{k+1}}{n_1}} \quad \text{for } 1 \leq r \leq k \tag{2.11}$$

When $c(w_{t-n+1:t}) > 0$, $P(w_t|w_{t-n+1:t-1}) = d_{c(w_{t-n+1:t})}\frac{c(w_{t-n+1:t})}{c(w_{t-n+1:t-1})}$. When an n-gram $w_{t-n+1:t}$ does not appear in the training set, the conditional probability $P(w_t|w_{t-n+1:t-1})$

depends on a lower order n-gram conditional distribution $P(w_t|w_{t-n+2:t-1})$, i.e.

$$P(w_t|w_{t-n+1:t-1}) = \alpha P(w_t|w_{t-n+2:t-1}) \tag{2.12}$$

where

$$\alpha = \alpha(w_{t-n+1:t-1}) = \frac{1-\sum_{w_t:c(w_{t-n+1:t})>0} P(w_m|w_{t-n+1:t-1})}{1-\sum_{w_m:c(w_{t-n+1:t})>0} P(w_t|w_{t-n+2:t-1})} \tag{2.13}$$

If $c(w_{t-n+1:t-1}) = 0$, then $P(w_t|w_{t-n+1:t-1}) = P(w_t|w_{t-n+2:t-1})$. Summarising all equations together, the conditional probability for w_t given $w_{t-n+1:t-1}$ can be written as

$$\begin{aligned} P(w_t|w_{t-n+1:t-1}) = d_{c(w_{t-n+1:t})}\frac{c(w_{t-n+1:t})}{c(w_{t-n+1:t-1})} + \\ (1-H(c(w_{t-n+1:t})))\alpha(w_{t-n+1:t-1})P(w_t|w_{t-n+2:t-1}) \end{aligned} \tag{2.14}$$

where

$$H(c(w_{t-n:t})) = \begin{cases} 1 & \text{if } c(w_{t-n:t}) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.15}$$

Since Katz smoothing only uses lower order n-gram probabilities when an n-gram is unseen, it is considered as a back-off model.

2.2.2 Kneser-Ney Smoothing

Kneser-Ney smoothing (Kneser and Ney, 1995) is an extension of absolute discounting (Ney et al., 1994). Using absolute discounting, the conditional probability of word w_t given previous $n-1$ words $w_{t-n:t-1}$ is

$$\begin{aligned} P_{abs}(w_t|w_{t-n+1:t-1}) = \frac{\max\{c(w_{t-n+1:t}) - D, 0\}}{\sum_{w_i} c(w_{t-n+1:t})} + \\ \frac{D}{\sum_{w_i} c(w_{t-n+1:t})} N_+(w_{t-n+1:t-1}\cdot) P_{abs}(w_t|w_{t-n+2:t-1}) \end{aligned} \tag{2.16}$$

where $N_+(w_{t-n+1:t-1}\cdot) = |\{w_t : C(w_{t-n+1:t-1} : w_t) > 0\}|$,

Some words only follow some specific words, say word y can appear after x. With absolute discounting, the smoothed probability may be too high if the uni-gram probability $p(y)$ is taken for backing-off. Kneser-Ney smoothing is able to back off to a low uni-gram probability in this case. An example given by Chen and Goodman (1998) indicates that `San Francisco` may appear many times in the training set, but `Francisco` can only occur

after `San`. Although the number of times `Francisco` appears in the training set is high, the uni-gram probability $P(\texttt{Francisco})$ should be low, since it only follows `San`. Therefore, the uni-gram probability of a word may be better to be proportional to the number of words that it follows, rather than proportional to the number of occurrence.

The conditional probability can be written in the following form:

$$P(w_t|w_{t-n+1:t-1}) = \begin{cases} \frac{\max\{c(w_{t-n+1:t})-D,0\}}{\sum_{w_i} c(w_{t-n+1:t})} & \text{if } c(w_{t-n+1:t}) > 0 \\ \gamma(w_{t-n+1:t-1})P(w_t|w_{t-n+2:t-1}) & \text{otherwise} \end{cases} \tag{2.17}$$

where $\gamma(w_{t-n+1:t-1})$ is the normalization term which guarantees $P(w_t|w_{t-n+1:t-1})$ sums to one. After some derivation, the term $P(w_t|w_{t-n+2:t-1})$ can be written in the following form:

$$P(w_t|w_{t-n+2:t-1}) = \frac{N_+(\cdot w_{t-n+2:t})}{N_+(\cdot w_{t-n+2:t-1}\cdot)} \tag{2.18}$$

where

$$N_+(\cdot w_{t-n+2:t}) = |w_{t-n+1} : c(w_{t-n+1:t}) > 0|$$
$$N_+(\cdot w_{t-n+2:t-1}\cdot) = |(w_{t-n+1}, w_t) : c(w_{t-n+1:t}) > 0|$$

2.3 Decoding Process

The ASR system finds the word sequence $\hat{W}$ such that:

$$\begin{aligned} \hat{W} &= \text{argmax}_W P(O|W)P(W) \\ &= \text{argmax}_W \left(\sum_X \sum_Q p(O,X|Q)P(Q|W)P(W) \right) \end{aligned} \tag{2.19}$$

where X is the set of all possible states, Q is the possible phone sequence given word sequence W and O is the acoustic vector sequence. $P(O,X|Q)$ can be computed by the acoustic model, $P(Q|W)$ is pronunciation probability and $P(W)$ is the prior from the language model. The summation over Q and X are computationally infeasible. Therefore, the Viterbi algorithm can be used in decoding process to find the approximation of Equation 2.19, which replaces the summation by maximisation. Based on the acoustic model, dictionary and language model, a finite state graph can be constructed and Viterbi decoding (Viterbi, 1967) can search this graph for finding the most probable state sequence.

2.4 Evaluation Metrics for Language Models

Two methods are considered for measuring the performance of different language models, namely perplexity and WER.

2.4.1 Perplexity (PPL)

Given a sequence of words $w_{1:t}$, the perplexity (PPL) is computed as:

$$\text{PPL} = 2^{-\frac{1}{t}\log_2 P(w_{1:t})} \tag{2.20}$$

The joint probability in Equation 2.20 can be rewritten as:

$$P(w_{1:t}) = \prod_{i=1}^{t} P(w_i|w_{1:i-1}) \tag{2.21}$$

where the decomposed conditional probabilities can be obtained by the language models. Therefore, for a dataset with N words, the language model PPL is

$$\text{PPL} = 2^{-\frac{1}{N}\sum_{i=1}^{N}\log_2 P(w_i|w_{1:i-1})} \tag{2.22}$$

A better language model should on average assign higher probabilities to the correct words, which means they should have a lower PPL. The PPL is a task independent metric for evaluating language models. When comparing different language models using PPL, it is necessary to use the same words in the output dictionary, since the language model with a smaller number of words in the dictionary can lead to lower PPL.

2.4.2 Word Error Rate (WER)

After aligning the reference transcription and the hypothesised word sequence, which minimises the Levenshtein distance, a common way to measure the performance of a LM in an ASR task is to find the word error rate (WER). The WER can be calculated by Equation 2.23.

$$\text{WER} = \frac{S+D+I}{N} \times 100\% \tag{2.23}$$

where S is the number of substitution errors, D is the number of deletion errors and I is the number of insertion errors.

Chapter 3

Neural Network Language Models

Traditional n-gram models often face the curse of dimensionality due to insufficient training data available. To better represent the relationship between words, a lot of research has focused on representing words in a continuous vector space. Typically examples are Continuous-Bag-Of-Words and Skip-Gram introduced by Mikolov et al. (2013a), which use a large amount of unstructured data to train a high quality vector representations of words. Words with similar meanings should be close to each other in the projected continuous vector space.

3.1 Feed-Forward Neural Network Language Model

The Feed-Forward Neural Network Language Model (FFLM) performs better than many previous language models (Goodman, 2001). It learns word feature vectors simultaneously with the parameters of the model (Bengio et al., 2003). The word feature vectors and the other parameters are tuned together in order to minimise the negative log-likelihood [1] of the training data. It is similar to n-gram model since it also uses the previous $n-1$ words as history. The difference is that the output probability of a n-gram FFLM model is conditioned on the feature vectors of the previous $n-1$ words.

The model structure of a 4-gram FFLM is shown in Figure 3.1. The inputs are the previous three words, which can be called context words.

The following steps describe the model structure in more details:

1. A shared matrix C maps the 1-of-K encoding of context words into a continuous vector space, i.e. Embedding(w) = C 1-of-K(w). Let $|V|$ be the dictionary size and m be the word feature size. Then $C \in \mathbb{R}^{m \times |V|}$. For simplicity Embedding(w) will be denoted

[1] equivalent to minimising perplexity

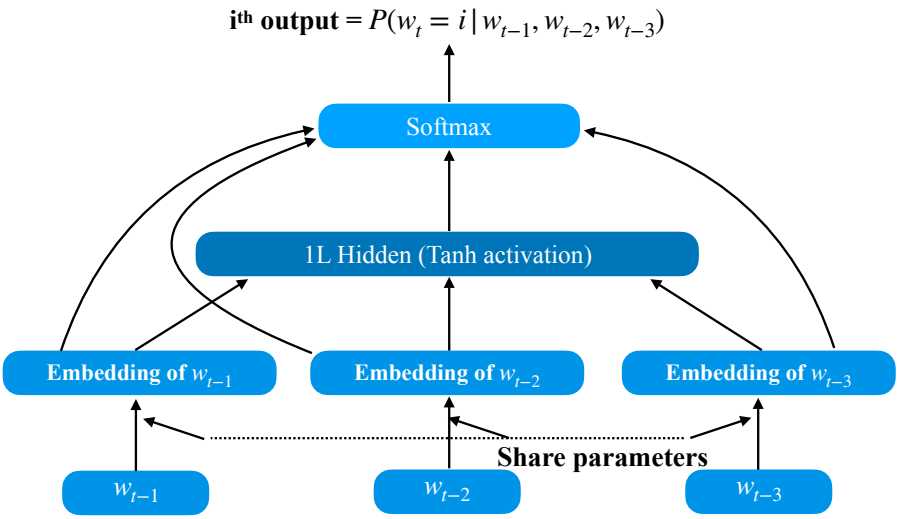


Fig. 3.1 4-gram FFLM structure

as $C(w)$. In an n-gram model the context words $w_{t-1},\cdots,w_{t-n+1}$ will be mapped to $C(w_{t-1}),\cdots,C(w_{t-n+1})$.

2. The concatenation of the context vectors forms the word features layer activation vector $\mathbf{x} = (C(w_{t-1}), C(w_{t-1}),\cdots,C(w_t - n + 1))$, where $\mathbf{x} \in \mathbb{R}^{(n-1)m}$.

3. If the hidden layer has h nodes, the un-normalised log probabilities for word w_i is computed as
$$\mathbf{y} = \mathbf{b} + \mathbf{W}x + \mathbf{U}\tanh(\mathbf{d} + \mathbf{H}\mathbf{x})$$
with parameters $\mathbf{b}, \mathbf{W}, \mathbf{U}$ and $\mathbf{H}$, where $\mathbf{b} \in \mathbb{R}^{|V|}$ is the output bias, $\mathbf{d} \in \mathbb{R}^h$ is the hidden layer biases, $\mathbf{W} \in \mathbb{R}^{|V|\times(n-1)m}$ helps to directly connect embeddings to the *softmax* output layer. $\mathbf{U} \in \mathbb{R}^{|V|\times h}$ is the weight matrix between the hidden and output layers and $\mathbf{H} \in \mathbb{R}^{h\times(n-1)m}$ is the weight matrix projecting word embeddings $\mathbf{x}$ to hidden layer.

4. The final output is then obtained by the softmax equation
$$P(w_t|w_{t-1}, w_{t-2},\cdots,w_{t-n+1}) = \frac{\exp y_{w_t}}{\sum_i \exp y_i}$$

Unlike traditional count-based n-gram models where the number of model parameters grows exponentially with the n-gram order, the number of parameters of the FFLM grows linearly when the order N increases.

3.2 Recurrent Neural Network Language Model

Even though the FFLM can mitigate the data sparsity problem by using word feature vectors, the context it can consider is still limited to the $n-1$ preceding words. The RNNLM (Mikolov et al., 2010) is one approach to consider much longer word history, i.e.

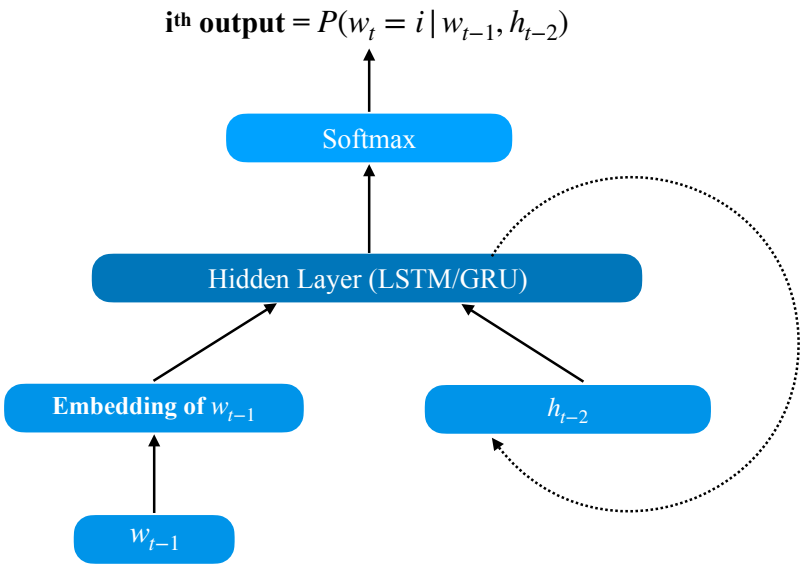


Fig. 3.2 Structure of RNNLM

As shown in Figure 3.2, an RNNLM only takes the word at time $t-1$ as input when predicting the word at time t. However, the output of the hidden layer from the previous time step (h_{t-2}) is concatenated with the embedding of the current input w_{t-1} to form the input to the hidden layer. This architecture gives the RNNLM the potential to consider contexts with unlimited size. However, there are several difficulties when trying to learn long-term dependencies with neural networks (Bengio et al., 1994).

One of the difficulties is the vanishing gradient problem. If the hidden layer is a feed-forward layer with a sigmoid activation function, the gradient could decay very quickly when it propagates back through the network.

3.2.1 LSTM

Long Short-Term Memory neural network architecture can mitigate the vanishing gradient problem (Sundermeyer et al., 2012). The memory cells have the ability to keep and forget history information.

With fully connected hidden layer, the hidden output h_{t-1} is computed as:

$$\mathbf{h}_{t-1} = \sigma(\mathbf{A}\mathbf{x}_{t-1} + \mathbf{B}\mathbf{h}_{t-2}) \tag{3.1}$$

where $\mathbf{x}_{t-1}$ is the feature vector of w_{t-1}, $\mathbf{h}_{t-2}$ is the hidden output from the previous step and $\mathbf{A}$, $\mathbf{B}$ are weights. σ is the sigmoid activation function.

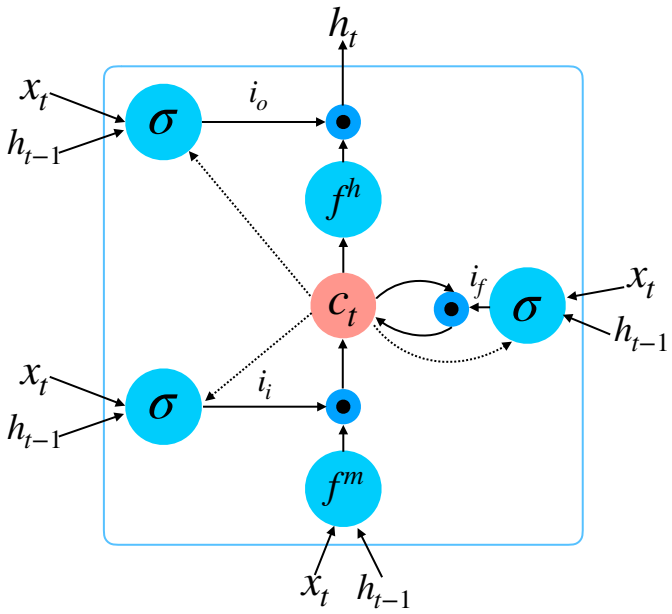


Fig. 3.3 LSTM memory cell with gating units.

After changing to an LSTM memory cell, the output of h_{t-1} requires more computation (shown in Figure 3.3):

$$\begin{aligned}
\mathbf{i}_i &= \sigma(\mathbf{W}_i^f \mathbf{x}_t + \mathbf{W}_i^r \mathbf{h}_{t-1} + \mathbf{W}_i^m \mathbf{c}_{t-1} + \mathbf{b}_i) \\
\mathbf{i}_f &= \sigma(\mathbf{W}_f^f \mathbf{x}_t + \mathbf{W}_f^r \mathbf{h}_{t-1} + \mathbf{W}_f^m \mathbf{c}_{t-1} + \mathbf{b}_f) \\
\mathbf{i}_o &= \sigma(\mathbf{W}_o^f \mathbf{x}_t + \mathbf{W}_o^r \mathbf{h}_{t-1} + \mathbf{W}_o^m \mathbf{c}_t + \mathbf{b}_o)
\end{aligned} \tag{3.2}$$

where $\mathbf{i}_i$, $\mathbf{i}_f$, $\mathbf{i}_o$ are input, forget and output gates respectively, $\mathbf{x}_t$ is the feature vector for word w_t, $\mathbf{W}$ are full weight matrices and $\mathbf{b}$ are biases. The memory cell $\mathbf{c}_t$ and the hidden layer output $\mathbf{h}_t$ are then calculated as:

$$\begin{aligned}
\mathbf{c}_t &= \mathbf{i}_f \odot \mathbf{c}_{t-1} + \mathbf{i}_i \odot f^m(\mathbf{W}_c^f x_t + \mathbf{W}_c^r \mathbf{h}_{t-1} + \mathbf{b}_c) \\
\mathbf{h}_t &= \mathbf{i}_o \odot f^h(\mathbf{c}_t)
\end{aligned} \tag{3.3}$$

where $\odot$ represents element-wise multiplication.

The memory cell **c** has a gated connection between consecutive steps. Also the current information $\mathbf{x}_t$ and past information $\mathbf{h}_{t-1}$ also contribute to the current memory cell $\mathbf{c}_t$. Since the current hidden state uses the output gate and an non-linear activation function to extract information from the current memory cell, older history is easier to be taken account of when predicting the current word.

3.2.2 GRU

The gated recurrent unit (Chung et al., 2014) can also be used to reduce the vanishing gradient problem. It requires fewer computation than an LSTM, which can be shown in the following equations:

$$\begin{aligned}
\mathbf{i}_f &= \sigma(\mathbf{W}_f^f \mathbf{x}_t + \mathbf{W}_f^r \mathbf{h}_{t-1} + \mathbf{b}_f) \\
\mathbf{i}_o &= \sigma(\mathbf{W}_o^f \mathbf{x}_t + \mathbf{W}_o^r \mathbf{h}_{t-1} + \mathbf{b}_o) \\
\hat{\mathbf{h}}_t &= f(\mathbf{W}_h^f \mathbf{x}_t + \mathbf{W}_h^r (\mathbf{i}_f \odot \mathbf{h}_{t-1}) + \mathbf{b}_h) \\
\mathbf{h}_t &= \mathbf{i}_o \odot \mathbf{h}_{t-1} + (\mathbf{1} - \mathbf{i}_o) \odot \hat{\mathbf{h}}_t
\end{aligned} \tag{3.4}$$

Comparing to the equations for LSTM, the equations for GRU do not include an output gate. Therefore, the number of parameters to update is smaller. Chung et al. (2014) shows that GRUs performs similar to LSTMs in many tasks.

3.2.3 Truncated Back Propagation Through Time

Back propagation through time (BPTT) (Werbos, 1990) is commonly used for training RNNLMs, which is an extension to the original back propagation. The errors can be traced back to the start of a data stream. Since an activation function such as sigmoid multiplies the gradient by a factor less than one when propagating back, the vanishing gradient problem is likely to appear after the gradient propagates back for too many steps. The truncated version of BPTT unfolds the RNNLM to a fixed number of steps (Mikolov et al., 2010). In this way, the model is updated more frequently and can prevent vanishing gradient to some extent.

3.3 Transformer Language Model

The attention mechanism was introduced in the context of machine translation in 2014 (Bahdanau et al., 2014). Bahdanau et al. (2014) proposed to use the attention weights for the encoder outputs from bi-directional recurrent neural network.

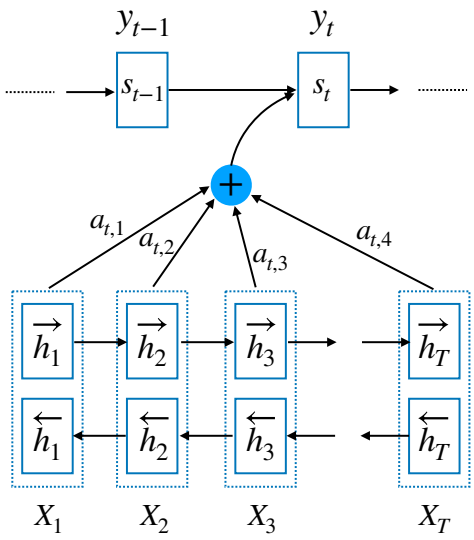


Fig. 3.4 Attention in neural machine translation sequence to sequence model

As shown in Figure 3.4, x_t is the source language input, y_t is the translated target language output, s_t is the RNN hidden state and the concatenation of forward state $\overrightarrow{h_t}$ and backward state $\overleftarrow{h_t}$ in the encoder part forms annotation vector h_t. The probability of each translated word given input sentence is computed as:

$$p(y_t|y_{1:t-1},\mathbf{x}) = g(y_{t-1},s_t,c_t)$$

where c_t is the context vector, which is the weighted sum of the annotations $h_{1:T}$:

$$c_i = \sum_{j=1}^{T} \alpha_{ij} h_j$$

and the weights α_{ij} is calculated as

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T}(e_{ik})}$$

where e_{ij} is the output of a feed-forward neural network alignment model which scores the relationship between h_j and s_{i-1}.

Transformer models also use an attention mechanism, but the attention weights are computed differently compared to the recurrent model. Both encoder and decoder parts in Transformer require attention weights.

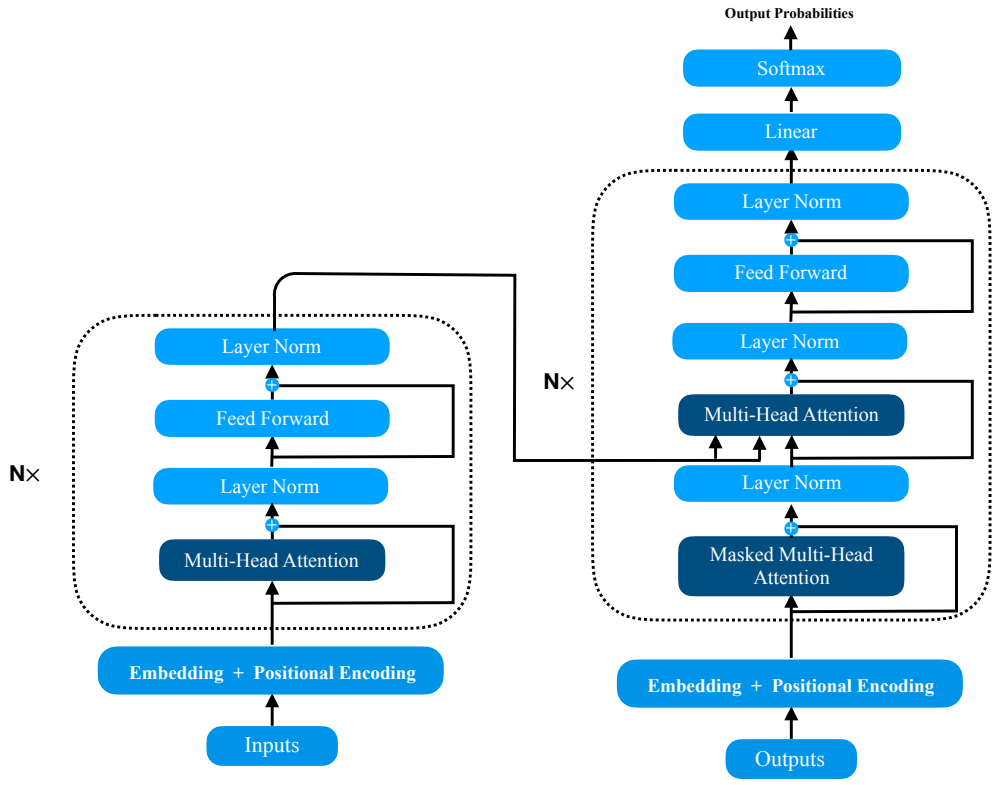


Fig. 3.5 Transformer for translation task

The attention function requires three inputs, namely query (**Q**), key (**K**) and value (**V**). Normally they are matrices of the same size. The output of the attention function is computed as:

$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}})\mathbf{V} \tag{3.5}$$

Multi-head attention can jointly attend to different representation subspaces at different locations, which is more powerful than a single attention (Vaswani et al., 2017). With head number h, it projects **Q**, **K** and **V** to dimension d_k with h times. The dimension d_k is chosen to be $\text{d}_{\text{model}}/h$ where d_{model} is the embedding size:

$$\text{MultiHead}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{Concat}(\text{head}_1,\ldots,\text{head}_h)\mathbf{W}^O \tag{3.6}$$

where $\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V)$ and $\mathbf{W}_i^Q$, $\mathbf{W}_i^K$ and $\mathbf{W}_i^V$ both have dimension $\mathbb{R}^{\text{d}_{\text{model}} \times d_k}$ and $\mathbf{W}^O \in \mathbb{R}^{\text{d}_{\text{model}} \times \text{d}_{\text{model}}}$. Since the dimension d_k is reduced as the number of heads increases, the computational cost is similar to finding attention with a single head.

The other common elements in both encoder and decoder are the `Add & Norm` layer, `Feed Forward` layer, and `Positional Encoding`.

The `Add & Norm` first adds a residual connection between hidden representation of words to the output from the multi-head attention layer. Then layer normalization (Ba et al., 2016) is applied to the total sum of the output. In order to use residual connections throughout the network, the output from each layer is the same as the embedding size.

The `Feed Forward` layer consists of two linear transformations with a ReLU activation in between:

$$\text{FFN}(\mathbf{x}) = \max(0, \mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2 \tag{3.7}$$

where $\mathbf{W}_1 \in \mathbb{R}^{\text{d}_{\text{model}} \times \text{d}_{\text{ff}}}$ and $\mathbf{W}_2 \in \mathbb{R}^{\text{d}_{\text{ff}} \times \text{d}_{\text{model}}}$ and d_ff is the size of the first linear transformation.

`Positional Encoding` injects some position information to the input sequence. It has the same size as the word embeddings so that they can be summed together. Vaswani et al. (2017) use fixed positional encoding as follows:

$$\text{PE}_{(\text{pos},2\text{i})} = \sin(\frac{\text{pos}}{10000^{2i/\text{d}_{\text{model}}}}) \tag{3.8}$$

$$\text{PE}_{(\text{pos},2\text{i}+1)} = \cos(\frac{\text{pos}}{10000^{2i/\text{d}_{\text{model}}}}) \tag{3.9}$$

Using Transformer structure for language modelling only requires one of the encoder or decoder parts from Transformer architecture shown in Figure 3.5. The training procedures and the task objectives are different for different styles. Both encoder style and decoder style Transformer language models will be reviewed in the following sections.

3.3.1 Decoder Style Transformer NNLM

Normally, the decoder style transformer LM is an autoregressive model, which does not allow the model to see future words when predicting the current word. It does not have the encoder part of the Transformer shown in Figure 3.5. Without taking the encoder output as the one of the inputs to the transformer layer, the structure of the decoder part can be simplified. It only consists of one `Mask Multi-Head Attention` with residual connections and one `Feed-Forward` sublayer with residual connections. Figure 3.6 shows the structure of the decoder style TLMs.

In decoder style TLM, a `right attention mask` is needed to make sure the model only sees the left context. This is because the TLM reads the sequence $w_{1:t-1}$ for predicting sequence $w_{2:t}$. The attention mask ensures that the model is a left-to-right language model.

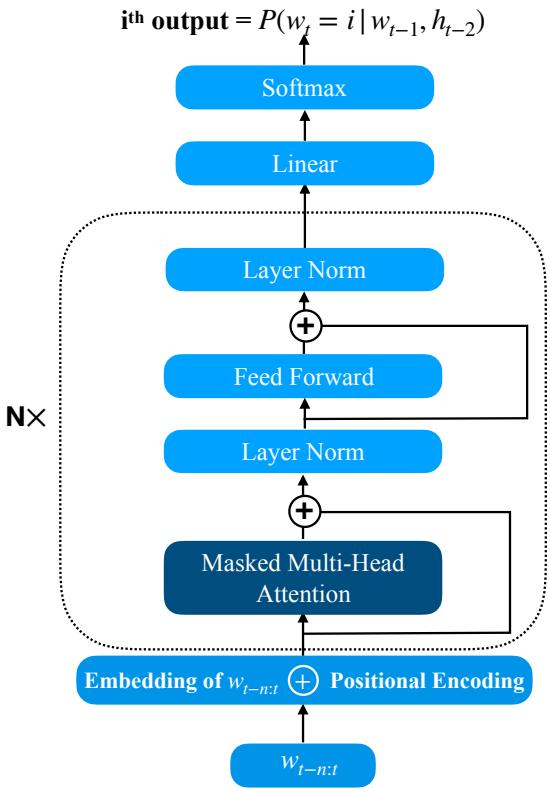


Fig. 3.6 decoder style TLM

If the model reads a sequence of four words (or subwords), the mask can be seen in Figure 3.7. Inputs $w_{1:4}$ are used to predict outputs $w_{2:5}$. When predicting word w_2, the model can only see the information from word w_1 and nothing else. And word w_3 is predicted base on words $w_{1:2}$ only. Without the mask, the information from word w_2 will leak and the training becomes trivial.

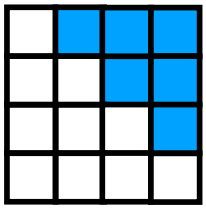
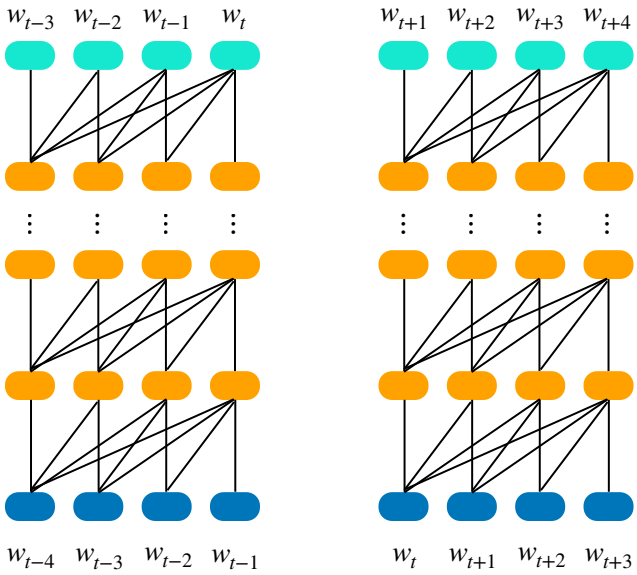


Fig. 3.7 `right attention mask` when the sequence length is four. Blue positions have attention score zero.

Each Transformer layer outputs a hidden representation of the context created by the words on the left of the word the model wants to predict. The training procedure for a decoder style transformer with sequence of four words is shown in Figure 3.8. The data stream will

be divided into disjoint groups, each group contains four words. For example, $w_{1:n}$ is divided into $\{[w_{1:4}], [w_{5:8}], [w_{9:12}], \dots\}$. Each group i is sent to the model in order in step i, and the parameters are updated for each step. When the mini-batch size is one, the RNNLM needs to compute $P(w_t|h_{t-1})$ first before finding $P(w_{t+1}|h_t)$ with the new input w_t. Finding only one probability distribution at a time makes the training speed slow. For a Transformer with sequence length four, the model can find the probabilities for four consecutive target words at once, which improves the training speed. Therefore, it would be sensible to train a deep transformer language model (Irie et al., 2019).



(a) training step i (b) training step $i+1$

Fig. 3.8 The forward procedure propagates from the bottom to the top. Each orange level represents the output hidden representation of a Transformer layer. The bottom level is the word embeddings plus positional encodings. The top level is the softmax to find the probability of each word given history. Solid lines show the paths with non-zero attention scores, which represent the dependencies between hidden representations in a level and hidden representations in the level above.

3.3.2 Encoder Style TLM

An encoder style TLM can predict the current word with bi-directional context (with both previous words and future words). The difference between the structures of encoder style and decoder style TLM is that the `Masked Multi-Head Attention` in Figure 3.6 not have attention mask now, which is called `Multi-Head Attention`.

However, simply removing the attention mask causes the information about current word to leak, i.e., the model now predicts $P(w_t|w_{1:t-1}, w_t, w_{t+n})$. To prevent seeing the current word that the model is to predict, approaches like BERT (Devlin et al., 2018), which mask the target word, are adopted.

If the objective of the encoder style TLM is to use previous n words and the future m words to predict the current word w_t, then the input to the model is $\{w_{t-n:t-1}, [\texttt{mask}], w_{t+1:t+m}\}$ and the output is $P(w_t|w_{t-n:t-1}, [\texttt{mask}], w_{t+1:t+m})$. Unlike the decoder style TLM, which gives n output distributions after taking n inputs. The encoder style TLM only gives one distribution for the target word which is changed to `[mask]` before sending to the model. Therefore, the training time can be much longer than decoder style Transformer NNLM.

3.4 Training Criterion for all NNLMs in this thesis

The cross entropy criterion is used for training all NNLMs in this thesis. The objective is to minimise the cross entropy loss. Given the softmax output and parameters θ, the cross entropy can be computed as,

$$\text{CE}(\theta) = -\sum_{w_j \in V} \tilde{P}(w_j|\text{context}) \log P(w_j|\text{context}, \theta) \tag{3.10}$$

where $\tilde{P}(w_j|\text{context})$ is the reference probability of word w_j in dictionary V, and $P(w_j|\text{context}, \theta)$ is softmax output from the given model. However, the cross entropy for bi-directional model cannot be easily converted to PPL, which is discussed in Section 5.6.2.

Since the dataset only contains the correct word, the probability $P(w_j|\text{context})$ is one when w_j is the correct word and zero otherwise. Therefore, the cross entropy can be estimated with training data of N words:

$$\text{CE}(\theta) = -\frac{1}{N}\sum_{i}^{N} \log P(w_i|\text{context}, \theta) \tag{3.11}$$

Note that $\exp(\text{CE}(\theta))$ is equivalent to PPL in Equation 2.22, so that minimising $\text{CE}(\theta)$ is the same as minimising PPL.

For generalisation purposes, the L2 regularization term is added to the loss function, which leads to the following equation to minimise:

$$\text{Loss}(\theta) = \text{CE}(\theta) + \frac{\lambda}{2N}\sum_{\theta} \theta^2 \tag{3.12}$$

where λ is the hyperparameter for L2.

3.5 Neural Network Language Models in ASR

Sometimes a NNLM can be too complex or costly to be applied to the search procedure in decoding process. Multi-pass decoding is often used as a solution (Young, 1996), where a lattice is generated first with an accurate acoustic model and a simple LM such as a bi-gram LM, and then rescored by a more complex LM. A lattice is a compact graph, where each node in a word lattice corresponds to a time instant, and the arcs connecting to a node contains word hypotheses with associated scores from acoustic model and language model. The lattice generated can be pruned so that the search space for the latter LM can be reduced.

After having a word lattice which is generated by the ASR system based on an acoustic model and an n-gram model, NNLMs are often used to rescore the n-gram model score. Lattice rescoring can be used if the model can be approximated to a finite-state model similar to an n-gram. Since for some LMs such as RNNLM and TLM which requires a long history context and the history representation in the lattice is complicated, n-best rescoring can also be a reasonable choice.

An n-best list is created by extracting the most likely paths in the lattice after interpolating an acoustic score and a n-best language model score. The score is defined as the log probability. The language model score of the n-best hypotheses for each utterance are then rescored by a NNLM and combined with the original acoustic scores. Often there is a language model scale factor γ which is tuned based on the dev set. The final score is computed by:

$$\text{Acoustic Score} + \gamma \times \text{LM Score} \tag{3.13}$$

The hypothesis with the highest final score after the language model rescoring process will then be selected and used to compute WER.

Chapter 4

Pre-trained Language Models

Using models such as skip-gram (Mikolov et al., 2013b) or GloVe (Pennington et al., 2014) to train the word embeddings with a large amount of data, and using the embeddings as the initialisation for the NNLMs has been found useful (Kim, 2014) in many NLP tasks. However, these methods only initialise the first layer of the NNLM. The remaining layers of the model still require effort to train. Another issue is that the embeddings do not incorporate contextual information. For example, The embedding for the word `bank` is the same for the investment bank and a river bank.

From 2018 onwards, pre-trained models such as Open AI GPT (Radford et al., 2018) or BERT (Devlin et al., 2018) have made the pre-trained methods more powerful. These new methods pre-train all layers in a deep model with a large amount of text from the Internet, and their final output hidden states can give more semantic information than previous methods.

Three pre-trained models will be reviewed, including their structures and the tasks on which they were pre-trained.

4.1 GPT

Open AI released its Generative Pre-Training (GPT) model in 2018. It uses the BooksCorpus dataset (Zhu et al., 2015) which contains over 7,000 unique unpublished books on a wide range of topics.

GPT uses the decoder style transformer structure mentioned in the previous chapter. Instead of using word level information, GPT uses a Byte Pair Encoding (BPE) (Sennrich et al., 2016) technique to transfer words into subword units. The vocabulary has 4,000 units.

The training procedure of this model can be split into two stages, namely the unsupervised pre-training stage and the supervised fine-tuning stage.

The first stage is unsupervised in the sense that the data are not labelled according to the discriminative task in the fine-tuning state and no matter what the task is, the first stage remains the same. The task in fine-tuning stage can be classification, semantic similarity and multiple choices. The GPT model contains 12 transformer layers, and 12 attention heads. The embeddings size is 768.

The second stage is fine-tuning stage. Generally in this stage the data is labelled so that it is called supervised. With input tokens $x_{1:m}$ and a corresponding label y, the fine-tuning objective becomes:

$$L_2(\theta) = \sum_{(x,y)} \log P(y|x_{1:m}) \tag{4.1}$$

For some tasks, there is improved performance by using auxiliary objective during fine-tuning stage, i.e.

$$L_3(\theta) = L_2(\theta) + \lambda * L_1(\theta) \tag{4.2}$$

where $L_1(\theta)$ is the unsupervised pre-training objective of the language model. In this project, the pre-training objective and the fine-tuning objective are both to predict the next token given the history.

In 2019, OpenAI released an updated version of GPT called GPT-2 (Radford et al., 2019), which is larger in size. The position of the Layer Norm is moved to the bottom of the `Feed Forward` sublayers.

4.2 Transformer XL

When using decoder style transformer language model reviewed above, the model only takes information from a fixed length context. The training is performed with a fixed-length segment of the pre-defined number of words or characters. Hence, the model may miss useful information when predicting the first few words or characters. For example, in Figure 3.8, predicting word w_{t+1} only uses word w_t during training.

Transformer-XL (Dai et al., 2019) takes the information from the previous training step to the next training step. In particular, instead of computing hidden outputs of each Transformer layer from scratch, Transformer-XL reuses the hidden outputs from the previous step. Reusing hidden states adds a recurrent connection between different steps. With these techniques, predicting w_{t+1} is able to see more history words.

For training step i the model uses segment $s_i = x_{i,1}, \ldots, x_{i,L}$ and step $i+1$ uses segment $s_{i+1} = x_{i+1,1}, \ldots, x_{i+1,L}$ where x is the character embeddings. Segments s_i and s_{i+1} are consecutive. Transformer-XL computes the n-th layer hidden state for segment s_{i+1} as follows:

$$\begin{aligned} \tilde{\mathbf{h}}_{i+1}^{n-1} &= \text{Concat}(\text{SG}(\mathbf{h}_i^{n-1}), \mathbf{h}_{i+1}^{n-1}) \\ \mathbf{q}_{i+1}^n, \mathbf{k}_{i+1}^n, \mathbf{v}_{i+1}^n &= \mathbf{h}_{i+1}^{n-1}\mathbf{W}_q, \tilde{\mathbf{h}}_{i+1}^{n-1}\mathbf{W}_k, \tilde{\mathbf{h}}_{i+1}^{n-1}\mathbf{W}_v \\ \mathbf{h}_{i+1}^n &= \text{Transformer} - \text{Layer}(\mathbf{q}_{i+1}^n, \mathbf{k}_{i+1}^n, \mathbf{v}_{i+1}^n) \end{aligned} \tag{4.3}$$

where $\text{SG}(\cdot)$ is a function to stop gradient backpropagation. $\mathbf{h}_i^{n-1}$ is the hidden representation in layer $n-1$ from the previous segment.

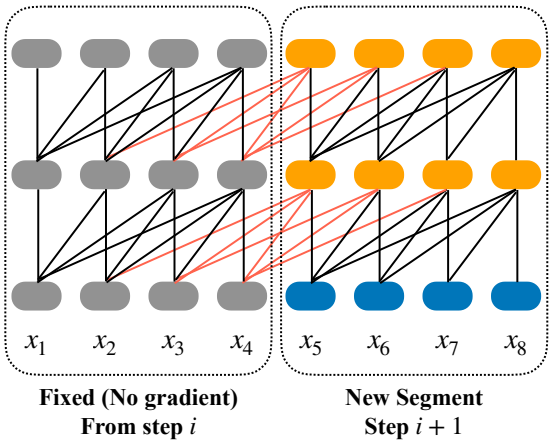


Fig. 4.1 Transformer-XL training procedure. Red Lines represents extra information the model receives from previous segment.

Another modification in Transformer-XL is that it uses relative positional encoding rather than absolute positional encoding. The hypothesis is that only the relative position is more important than absolute position. The relative positional matrix is a sinusoid matrix and is added when calculating the attentions.

4.3 BERT

BERT stands for Bidirectional Encoder Representations from Transformers (Devlin et al., 2018). The first significant difference between BERT and previous two pre-trained models is that it uses the encoder part of the standard Transformer (Vaswani et al., 2017). This means when calculating the attention, there is no attention mask involved. For predicting word w_t the model has the ability to see both the left and the right context. The pre-trained model

used in this project is $\text{BERT}_{\text{BASE}}$, which has exactly the same size as the GPT. BooksCorpus (Zhu et al., 2015) and English Wikipedia are used to pre-train BERT.

To prevent the model from seeing the current word, BERT replaces the word it is going to predict with a special symbol `[mask]`. For example, if a tokenised sequence is $\{x_1, x_2, x_3, x_4\}$ and the objective is to predict $P(x_3|x_1, x_2, x_4)$. BERT changes the original sequence to $\{x_1, x_2,$ `[mask]`, $x_4\}$, so that without any mask the model is able to predict the current token with bi-directional context.

The pre-trained model is trained on two tasks. One is the called `Mask LM`. The goal is to predict 15% of subwords in each sentence. Since some tasks do not have the special symbol `[mask]` during fine-tuning, among the 15% target subwords, 80% of them are replaced by `[mask]` before sending to the model, 10% are replaced by a random subword, and the final 10% stay the same. The goal of the second task is to classify if the two input sentences are consecutive and they are in the right order.

Chapter 5

Experiments

5.1 Setup

5.1.1 Data

The dataset used in this project is the Augmented Multi-party Interaction (AMI) corpus (Carletta et al., 2006), which consists of 100 hours of recorded meetings. The recordings in AMI are synchronised and collected in special instrumented meeting rooms. High quality transcriptions are generated as part of the corpus. This corpus was created because meetings are fairly common in daily life. Moreover, the discussions in meetings often need to be written down for documentation to ease the search in the future. However, it may be expensive to write down everything in the meeting manually. Thus, an accurate ASR system would be helpful in this situation.

The reason for choosing the AMI corpus in the project is that the AMI meeting transcription task is a standard ASR task. It also has a reasonably small training set, which allows more experiments to be conducted efficiently. The vocabulary used for this corpus contains 13077 words. After normalisation, the text data is summarised in Table 5.1.

Data set	Train	Dev	Eval
# words	911K	108K	102K
Duration (hours)	80	10	10

Table 5.1 Summary of AMI dataset

The same corpus has also been used in Sun's MEng thesis (Sun, 2019). The final results in WER can be compared to the results in his work directly.

5.1.2 ASR System For Generating 50-Best list

The acoustic model in the ASR system is a context-dependent TDNN model containing 4 ResNet blocks. The input feature vector is a mel-scaled log filer bank (FBANK) vector with first order differential, which has 80 dimensions. All hidden layers in this model uses 1000 hidden nodes and the output is a distribution of 8435 tri-phone states. Minimum Phone-Error-Rate (Povey and Woodland, 2002) is used as the training criterion. The 50-best list for LM rescoring in this thesis is extracted by the scores from the TDNN acoustic model and a tri-gram LM trained on AMI and Fisher corpora. HTK 3.5.1 and PyHTK (Zhang et al., 2019) are used to implement this ASR system.

5.2 Resources for Experiments

5.2.1 Scripts

For the FFLM, the code is written from scratch based on the implementation details shown by Bengio et al. (2003). The code for TLM can be used to train both decoder-style TLM and encoder-style TLM. The structure of decoder style TLM is based on Vaswani et al. (2017) by removing the components connected to the encoder part of the original Transformer since it was used for machine translation. The training procedure for encoder style TLM is influenced by the Mask LM task described in Devlin et al. (2018). However, the Mask LM task was modified in this project so that every word in the dataset is masked once and each of the masked word is given an output distribution. Every NNLM trained only on AMI corpus can be either a forward or a backward model.

Unlike other types of NNLMs which do not use any source code, the RNNLM extended the source code provided by Guangzhi Sun[1]. The provided code is a forward RNNLM, so it is later extended to a backward NNLM and the output format is modified in order to combine with other NNLMs.

There are two main codes for combining different NNLMs. The EM algorithm used for finding weights to compute combined PPL does not rely on any other source code, whilst the CMA-ES (will be described later) for finding weights that directly minimises WER are based on the example script given by Qiujia Li[2].

All scripts used for this thesis can be found on Github[3].

[1]Final year MEng student at Cambridge; Github link: https://github.com/BriansIDP/CULM
[2]First year PhD in Engineering at Cambridge
[3]https://github.com/Xianrui-Zheng/NNLM

5.2.2 Packages

The n-gram model trained on AMI corpus for testing PPL uses SRILM[4] package. All codes for training NNLMs are written using the Pytorch deep learning Python package. Other main Python packages used are `cma`, `matplotlib` and `Pandas`. Pre-train models are downloaded from `pytorch_transformers` package. All NNLMs presented in this section were trained on a single Tesla K80 GPU via Microsoft Azure [5]. The NIST Sclite scoring tool is used for finding WER in this project.

5.3 Experiments for individual NNLMs

This section introduces the data arrangement for each NNLM trained from scratch, together with some PPL results. WER results can be found in the next section when different NNLMs are compared with each other.

Given a dataset, each line is a sentence without sentence boundary symbol. For each dataset, the first step for all NNLM to process the data is to add a sentence separator (special symbol <eos>) at the end of each line, and then all lines are joined together to form a single string. The output is a list which contains all words (including <eos>) from the string in order. This list will be referred to as data list.

5.3.1 Experiments for FFLM

In order to predict the first word, n-1 `<eos>` are added to the beginning of the data list to serve as the context for the first word. Then a context window slices through the data list to find all possible n-grams, regardless whether they repeat or not. Then the n-grams are shuffled. Mini-batches are created in order to improve the efficiency of the training procedure when using the GPU. The mini-batch size is the number of n-grams fed into the model together. For example, if the data list contains the following elements in order:

$$[w_1, w_2, w_3, \text{<eos>}, w_4, w_5, w_6, w_7, \text{<eos>}]$$

Then all possible lists of tri-gram are:

$$[\text{<eos>}, \text{<eos>}, w_1]\ [\text{<eos>}, w_1, w_2]\ [w_1, w_2, w_3]$$

$$[w_2, w_3, \text{<eos>}]\ [w_3, \text{<eos>}, w_4]\ [\text{<eos>}, w_4, w_5]$$

$$[w_4, w_5, w_6]\ [w_5, w_6, w_7]\ [w_6, w_7, \text{<eos>}]$$

[4] http://www.speech.sri.com/projects/srilm/
[5] https://azure.microsoft.com/

where the first two elements in each list is the context, and the last element is the target to be predicted. If the mini-batch size is three, then at each time step during training procedure, three tri-grams are selected randomly without replacement. Shuffling is only done for training and the tri-grams will be selected for evaluation. Among the above tri-grams, if w_2 is the same as w_6 and w_3 is the same as w_7, then there would be at least two tri-grams that are identical after slicing the context window. Therefore, if three words often occur in consecutive order in a dataset, they would be sent to the model more frequently than other tri-grams.

Adding the direct connection between the linear transformation of embeddings and the output of the hidden layer helps to improve the performance:

Connect	N-gram	nhid	Mins	Train PPL	Dev PPL	Eval PPL
	3	256	27.89	54.48	81.43	74.07
✓	3	256	51.17	56.77	81.09	73.94

Table 5.2 tri-gram FFLM with 1 hidden layer. Connect represents direct connection, nhid is the number of hidden units in a feed-forward hidden layer and Mins is the training time in minutes with a single K80 GPU.

All other FFLM experiments would have the direct connection. The next experiment is to see if adding one extra hidden layer helps. Although the training PPL drops, both the dev PPL and eval PPL increase.

# Layer	N-gram	nhid	Mins	Train PPL	Dev PPL	Eval PPL
2	3	256	52.16	56.02	83.63	76.30

Table 5.3 Tri-gram FFLM with 2 hidden layers, the output size of the first hidden layer is 256.

With the one layer FFLM, the effect of using different n-gram is investigated:

The best result from all FFLM uses 5-gram and having more order leads to over-fitting problem. Fine-tuning the regularisation terms may be able to further improve the performance.

# Layer	N-gram	nhid	Mins	Train PPL	Dev PPL	Eval PPL
1	5	256	70.57	52.97	79.94	72.53

Table 5.4 5-gram right-to-left FFLM

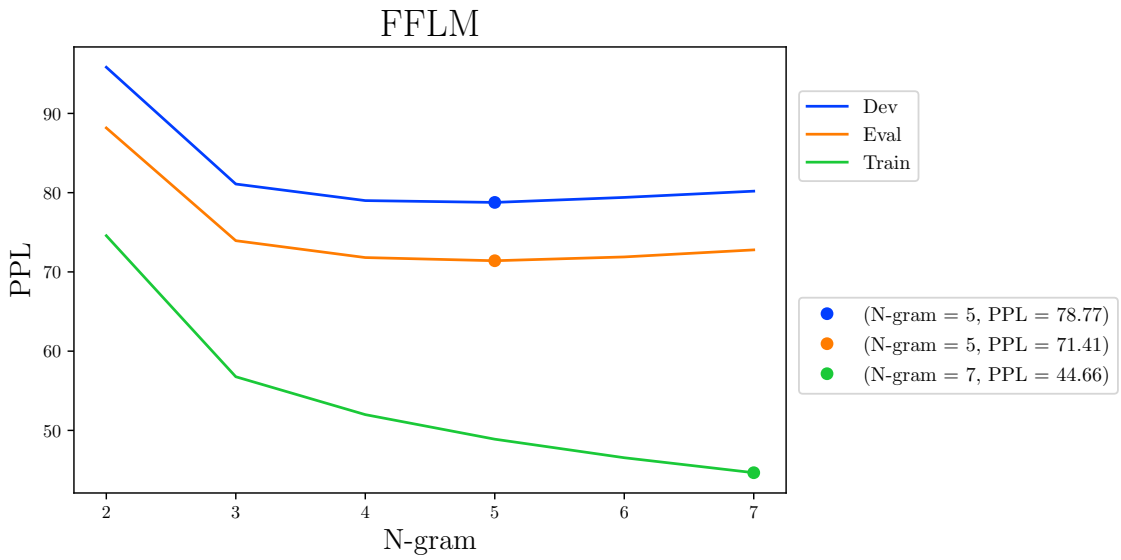


Fig. 5.1 1 layer FFLM with 256 hidden layer size. The points indicate the lowest PPLs

5.3.2 Experiment for RNNLM

Since RNNLM requires a continuous sequence of words, no shuffling is done to the data list. In order to predict the first word at the beginning of the data, an extra <eos> symbol is added to provide the required input for the model.

Mini-batches are created differently for RNNLM compared to FFLM. For predicting the first word with RNNLM, one <eos> symbol is added to the beginning of the data list. If the length of the data list is set to L, and the pre-defined mini-batch size is m such that $m > 1$, the list is trimmed so that the new list has length $\hat{L}$ which is an integer multiple of m. The maximum number of words from the dataset to be trimmed is $m - 1$ words. The next step is to form m streams of length $\frac{\hat{L}}{m}$. An example could be helpful, if the mini-batch size is three and a data list is the following:

$$[w_1, \text{<eos>}, w_2, w_3, \text{<eos>}, w_4, w_5, w_6, w_7, \text{<eos>}]$$

which has length 11 after adding an extra <eos> at the beginning, the data list will then be trimmed and has length nine (which is a integer multiple of mini-batch size three).

$$[\text{<eos>}, w_1, \text{<eos>}, w_2, w_3, \text{<eos>}, w_4, w_5, w_6]$$

The three columns form three streams, and each row is a mini-batch for RNNLM

<eos>	w_2	w_4
w_1	w_3	w_5
<eos>	<eos>	w_6

which are fed into the RNNLM model in order. When row $i \in \{1,2\}$ serves as input, row $i+1$ is the target prediction. Assuming the BPTT step size is k, the gradient will accumulate for k mini-batches to be sent to the model until it updates the parameters. The reason to keep the words in each stream in order is that consecutive words from each stream are required to be sent to the RNNLM.

During training time, some data is missing due to the trimming, while no trimming is required for evaluation time since the mini-batch size in evaluation procedure is always one. Therefore, all words in a dataset can be predicted during evaluation time.

Two types of RNNLM are examined, one uses GRUs and the other one uses an LSTM layer. As shown in Table 5.5, the LSTM has better performance and the later experiments will continue with LSTM only. More LSTM layers are considered, but no improvement is found.

Type	Mins	Train PPL	Dev PPL	Eval PPL
GRU	27.19	36.74	64.57	57.34
LSTM	31.03	29.01	63.97	56.55

Table 5.5 Comparison between 1 layer LSTM and 1 layer GRU with the same hyper-parameters. BPTT is set to 12, hidden unit size is 768 and embedding size is 256

The reason for using truncated BPTT is that the vanishing gradient problem can be further mitigated. Varying the step size for BPTT also influences the performance of RNNLM.

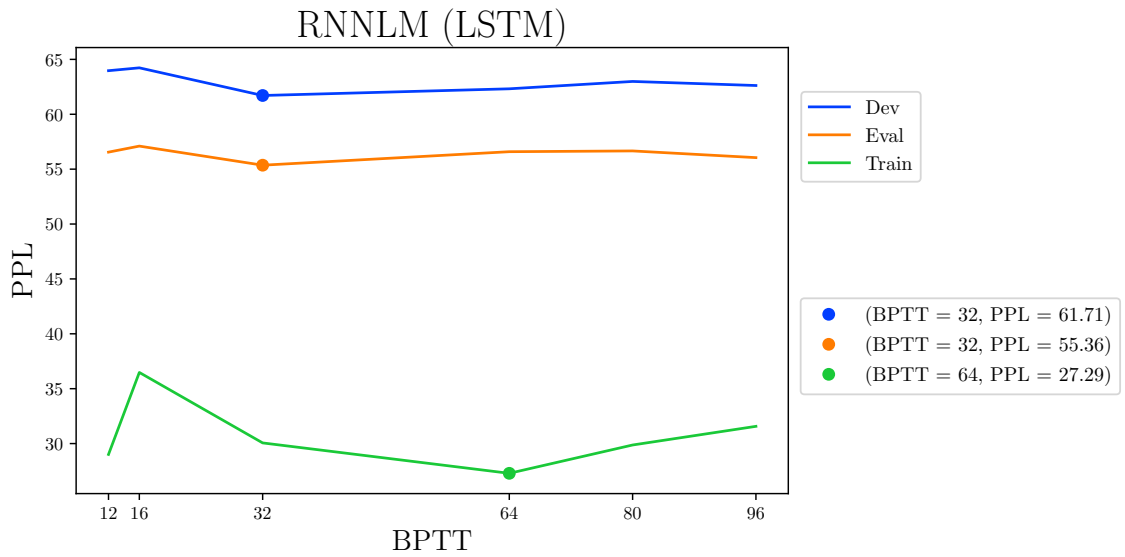


Fig. 5.2 1 layer LSTM with 768 hidden units. The points indicate the lowest PPLs

5.3.3 Experiments for TLM

Unless specified, all experiments for the TLM uses decoder style TLM.

There are three possible ways to arrange training data for the TLM, which are discussed as follows:

Method One

Method one is similar to the method used for training RNNLM. The difference is that each batch contains not only one word from each stream, but also a word sequence with pre-defined sequence length. With the same example for the RNNLM and the sequence length set to 2, the mini-batch changes to

<eos>	w_2	w_4
w_1	w_3	w_5
<eos>	<eos>	w_6

<eos>	w_2	w_4
w_1	w_3	w_5
<eos>	<eos>	w_6

where the blue square indicates the input mini-batch and the red square indicates the output targets. For stream two, the LM uses w_2 to predict w_3 and uses both w_2 and w_3 to predict <eos>.

Method Two

Since consecutive segments do not need to link together, shuffling can be done. After finding all input segments within each stream, which is packed with the target segments, the order of those packs can change within or across streams.

Method Three

Method one and two both split the stream into disjoint segments during training time. The consequence is that for each mini-batch, the first target word in each segment is predicted given only its previous word. If the training procedure changes to the method shown in Figure 5.3, all target words would have the contexts of the same length.

The segments are no longer disjoint within each stream, since the second word in the previous segment becomes the first word in the second segment and the third word in the previous segment becomes the second word in the first segemnt etc. This is also the way Transformer use to give prediction at evaluation time.

The PPL tested by all three datasets in AMI corpus are nearly the same with method one and two. When using method three to train the model, the training procedure takes much

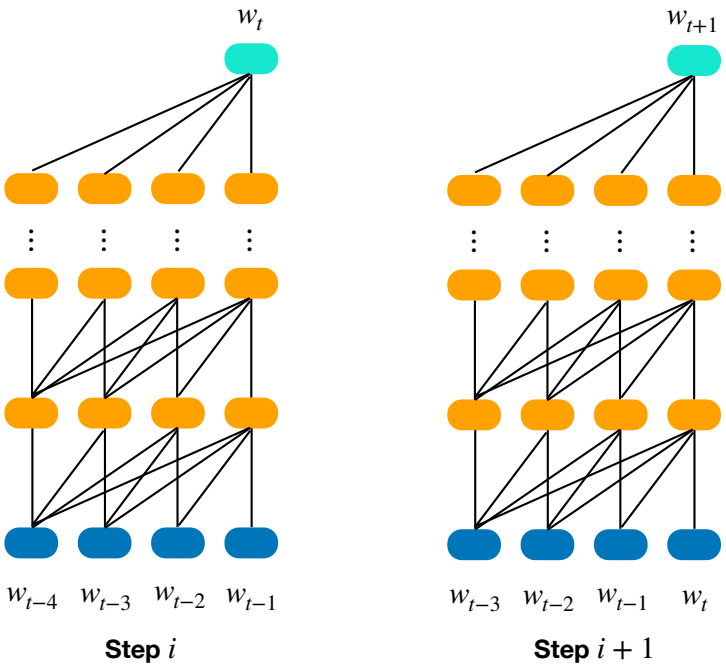


Fig. 5.3 Predict each word with the context of the same length

longer to complete. By considering the trade-off between using augmented data to train the model and efficiency, later experiments only use method one during training.

Experiments show that individually pre-trained Transformer layers help to speed up convergence. Each layer is pre-trained before adding subsequent layers on its top. If the network has n layers, for each $i \in [1, \ldots, n-1]$, layer i is trained for 1 epoch and then another layer will be added on the top of layer i.

For a TLM with one layer only, increasing sequence length does not improve the model performance, as shown in Figure 5.4:

However, the performance is better with longer sequence lengths when the model is deep. One possible explanation can be that the model would be better at using the history information if it is deep. As shown in Figure 5.5, the PPL drops rapidly when the number of layers increases from one to four. The downside with a deep model is that the training time becomes significantly longer than models with only one or two layers. In decoder-style TLM experiments, the layer norm and positional encoding are not found useful. One plausible hypothesis is that the positional encoding is useful only when the sequence length is set to be a much larger number (512 was set in (Dai et al., 2019; Devlin et al., 2018; Radford et al., 2018), but these models were trained on much larger datasets). However, for encoder style TLM, further experiments show that the positional encoding is necessary even when

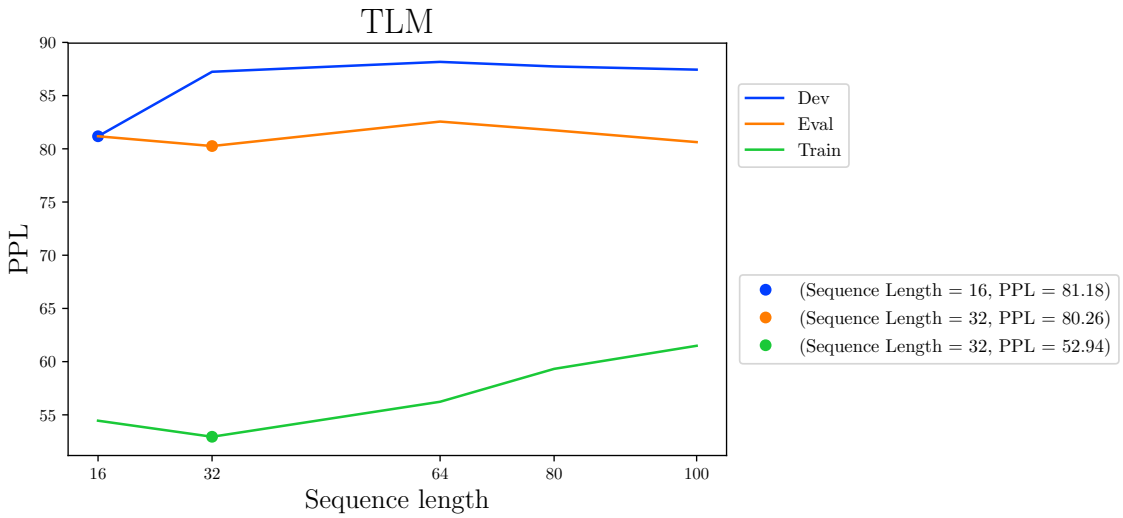


Fig. 5.4 1 layer decoder style Transformer with various sequence length

the sequence length is small. Since the encoder style TLM is a bi-directional model, extra encoding is needed to distinguish which direction a word embedding comes from.

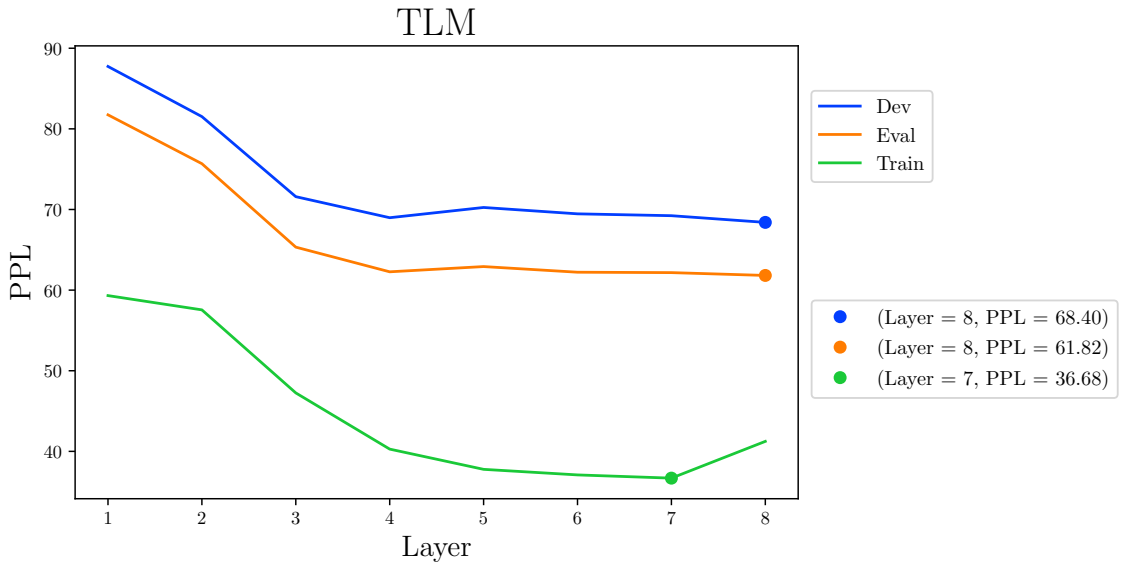


Fig. 5.5 Decoder style transformer with sequence length set to 80 words.

5.4 Comparison Between Different Individual LMs

5.4.1 Differences of PPL

Model	Direction	Mins	Train PPL	Dev PPL	Eval PPL
tri-gram-s	=>	-	44.67	91.08	81.53
FFLM	=>	71.89	48.89	78.77	71.41
FFLM	<=	70.57	52.97	79.94	72.53
RNNLM	=>	51.24	30.06	61.71	55.36
RNNLM	<=	51.21	31.01	62.17	55.21
TLM	=>	118.39	37.26	66.37	59.63
TLM	<=	116.47	40.60	69.52	61.48

Table 5.6 Comparison of the best result from each model; => represents forward model and <= represents backward model; Mins indicates the total length of training time in minutes; FFLM: 1 hidden layer with 256 hidden units; RNNLM: 1 layer LSTM with 32 BPTT steps and 768 hidden units; tri-gram-s: tri-gram model trained on AMI corpus. Transformer: 8 layers without positional encoding and layernorm, 8 attention heads, 768 hidden units for linear layers

Forward LMs are left-to-right models and backward LMs are right-to-left models. For backward LMs, the data list from each dataset is reversed before turning them into mini-batches. The difference of PPL between forward and backward NNLMs of the same type is much smaller than that between different types of NNLMs. For tri-gram model, Kneser-Ney Smoothing is used since it works better than other smoothing techniques (Goodman, 2001).

Among all forward and backward models, RNNLM performs the best in terms of PPL, but decoder style Transformer still has the potential to improve its performance by increasing the sequence length. However, the training time for Transformer is the longest since it has more layers than the other two NNLMs.

5.4.2 Variation of Log Probabilities

Forward Models

By randomly selecting an example of 11 consecutive words from the dev set, Figure 5.6 shows the $\log_2$ probabilities of those words predicted by different forward language models.

All models give high probabilities to words that appear frequently in the training set, such as <eos>, OF and THE. The overall fluctuation from different LMs seems to be similar, but there are some local variations. Every model provides the highest $\log_2$ probabilities for

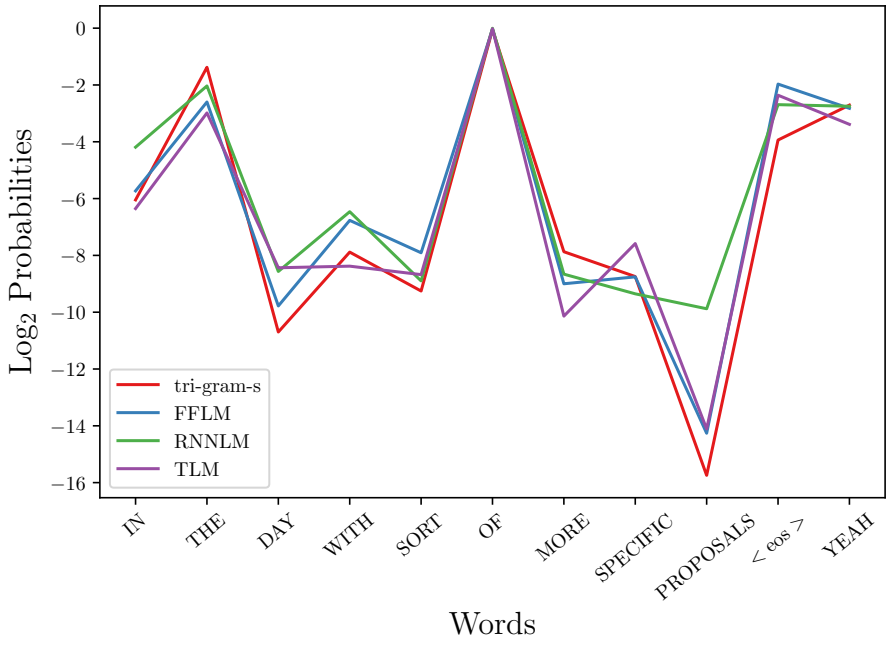


Fig. 5.6 the $\log_2$ probabilities of 11 consecutive words randomly selected from dev set

some words. In particular, RNNLM gives considerably higher $\log_2$ probability for word PROPOSALS. Therefore, interpolating their outputs would be a sensible way to reduce the PPL.

Backward Models

The variation among backward models is generally bigger than that among forward models. An example is shown in Figure 5.7, the trend in the middle for the three NNLMs are quite different, especially for Transformer. Similar observations are also found with other samples of consecutive words.

Based on Figures 5.6 and 5.7, it is clear that all models considered can predict words better than the others in some occasion. The next section will discuss possible combination methods.

5.5 Combination of Different LMs

This project uses the interpolation method to combine many language models. Each model will be used for n-best LM rescoring for speech recognition, and the outputs are linearly combined in order to find the 1-best result. Different ways are considered for finding the interpolation weights.

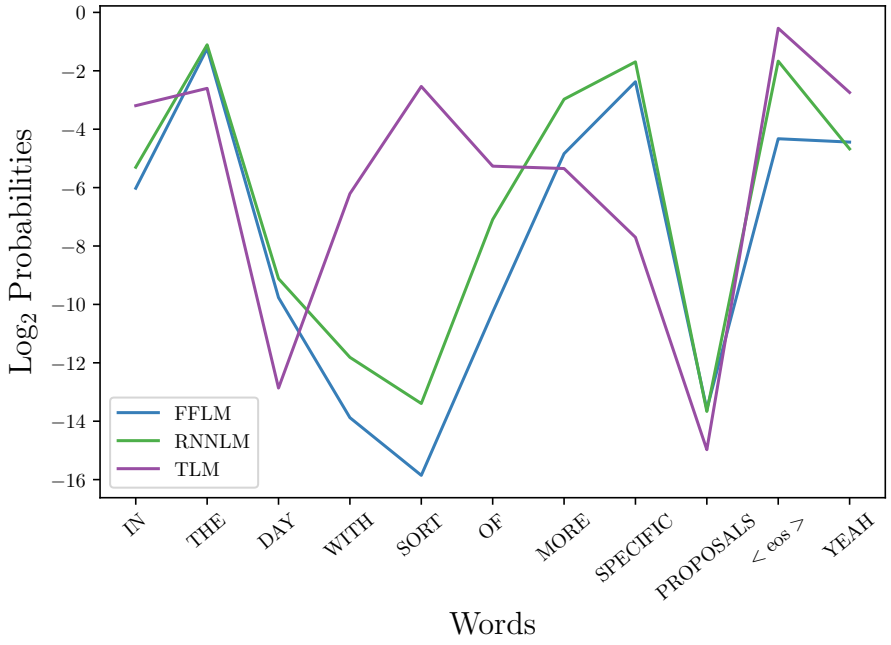


Fig. 5.7 the $\log_2$ probabilities of the same 11 consecutive words in Figure 5.6, but predicted by backward models

5.5.1 Minimize Dev PPL by Combining Word Probabilities

Each forward language model will predict the probability of each word given history. Assuming there are K models, using the conditional probabilities of all words in dev set, the combined probabilities of each word can be written as:

$$P(w_i|w_{1:i-1}) = \sum_{j=1}^{K} \lambda_j P(w_i|w_{1:i-1};m_j) \tag{5.1}$$

where m_j represents model j and λ_j is the interpolation weight for model j, then:

$$\begin{aligned} P(w_{1:N}) &= \prod_{i=1}^{N} P(w_i|w_{1:i-1}) \\ \implies \log P(w_{1:N}) &= \sum_{i=1}^{N} \log P(w_i|w_{1:i-1}) \\ &= \sum_{i=1}^{N} \log\{\sum_{j=1}^{K} \lambda_j P(w_i|w_{1:i-1};m_j)\} \end{aligned} \tag{5.2}$$

The objective is to maximise $\log P(w_{1:N})$ with respect to λ_j. Since $\sum_{j=1}^{K} \lambda_j = 1$, the Lagrange multiplier L is used to add this constraint. The updated objective with the constraint applied becomes the following:

$$
\begin{aligned}
y &= \sum_{i=1}^{N} \log\{\sum_{j=1}^{K} \lambda_j P(w_i|w_{1:i-1};m_j)\} - L(\sum_{j=1}^{K} \lambda_j) \\
\Longrightarrow \frac{\partial y}{\partial \lambda_j} &= \sum_{i=1}^{N} \frac{P(w_i|w_{1:i-1};m_j)}{\sum_{j=1}^{K} \lambda_j P(w_i|w_{1:i-1};m_j)} - L = 0 \\
\Longrightarrow \lambda_j L &= \sum_{i=1}^{N} \frac{P(w_i|w_{1:i-1};m_j)}{\sum_{j=1}^{K} \lambda_j P(w_i|w_{1:i-1};m_j)} \\
\Longrightarrow L &= N \\
\Longrightarrow \sum_{i=1}^{N} \frac{P(w_i|w_{1:i-1};m_j)}{\sum_{j=1}^{K} \lambda_j P(w_i|w_{1:i-1};m_j)} &= N \\
\Longrightarrow \lambda_j^{(t+1)} &= \frac{1}{N} \sum_{i=1}^{N} \frac{\lambda_j^{(t)} P(w_i|w_{1:i-1};m_j)}{\sum_{\ell=1}^{K} \lambda_\ell^{(t)} P(w_i|w_{1:i-1};m_\ell)}
\end{aligned}
\tag{5.3}
$$

This is an iterative process, which is called EM algorithm (Dempster et al., 1977). Since the goal is to maximise the log joint probability in Equation 5.2, the stopping criterion can be when the increase in log joint probability is less than 10^{-6}.

Combination of Forward Models

Based on Equation 5.3, the combination of all forward models is straightforward. The initial weights are set to $\frac{1}{\text{number of models}}$ for every model. The EM algorithm is applied to find the weights for each model such that the combined dev PPL is minimised, and then these weights are applied to eval set to find the combined eval PPL. As shown in Table 5.7, RNNLM and TLM get most of the weight, but adding FFLM and tri-gram to the combination still helps to further reduce the PPL.

Combine Forward and Backward Models

The joint probability of all words in a dataset can be decomposed into a product of all forward probabilities or all backward probabilities, i.e.

$$
P(w_{1:N}) = \prod_{i=1}^{N} P(w_i|w_{1:i-1}) = \prod_{i=1}^{N} P(w_i|w_{i+1:N}) \tag{5.4}
$$

Models Combined	Dev PPL	Eval PPL
Baseline	61.71	55.36
RNNLM + TLM	58.30	52.46
FFLM + RNNLM + TLM	58.09	52.31
FFLM + RNNLM + TLM + tri-gram-s	57.75	51.99

Table 5.7 Combination of forward models. Weights for the first line are 0.613, 0.389, weights for the second line are 0.084, 0.559, 0.357 and weights for the third line are 0.039, 0.545 0.342, 0.075. Baseline is the best single model without combination (RNNLM).

When trying to combine backward probabilities and forward probabilities, Equation 5.3 may no longer be valid. For example, when there is one forward model and one backward model, the combination cannot by done by simply interpolating the forward probability and the backward probability of each word, i.e.:

$$\log P(w_{1:N}) \neq \sum_{i=1}^{N} \log(\lambda_1 P(w_i|w_{1:i-1};m_1) + \lambda_2 P(w_i|w_{i+1:N};m_2)) \tag{5.5}$$

This would result in a non-valid log joint probability, which leads to a non-valid PPL that is much lower than the PPL for each model before combination.

Model	non-valid Dev PPL	non-valid Eval PPL
FFLM	24.32	23.47
RNNLM	19.13	18.06
TLM	38.65	33.62

Table 5.8 Combining the forward and backward models of the same type in Table 5.6

The average of non-valid log probabilities for each word after interpolating forward and backward models of the same type is higher than the average of the combination of all the forward models. For the same 11 words selected for plotting Figures 5.6 and 5.7, the non-valid $\log_2$ probabilities are also higher on average.

5.5.2 Minimize Dev PPL by Combining Sentence Probabilities

Instead of combining $\log_2$ probabilities of words, it is possible to combine $\log_2$ probabilities of sentences. A further assumption needs to be made, which says that the words in the current sentence do not relate to any word in other sentences. With this extra assumption, the joint

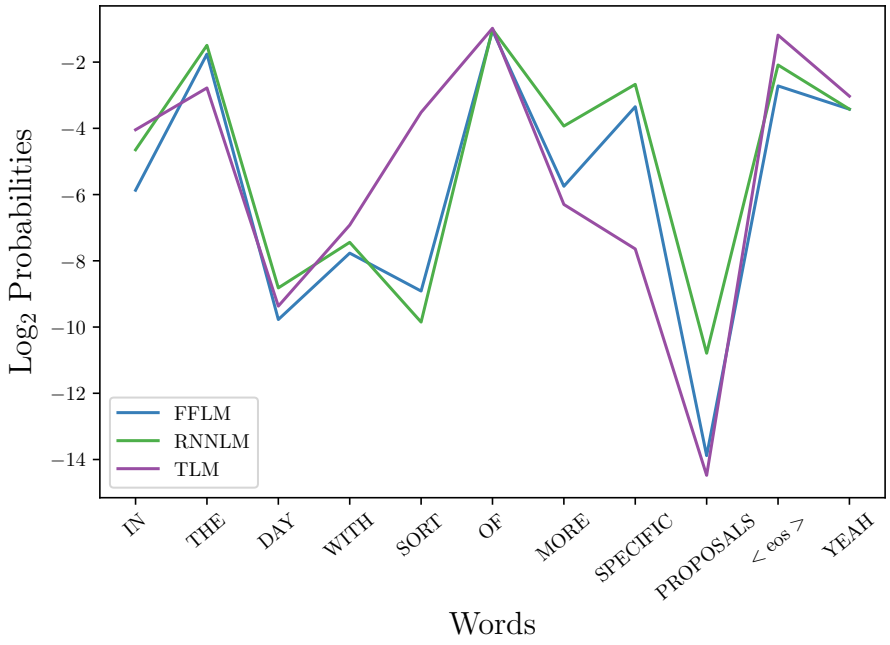


Fig. 5.8 Non-valid $\log_2$ probabilites of 11 words

probability of all words in a dataset is equal to the products of the sentence probabilities. If there are N words and Z sentences in a dataset, then

$$P(w_{1:N}) = \prod_{i=1}^{Z} P(s^i) \tag{5.6}$$

where s^i indicates sentence i. Each sentence probability $P(s^i)$ is computed by forward model as

$$P(s^i) = \prod_{j=1}^{\text{length}(s^i)} P(s^i_j | s^i_{1:j-1}) \tag{5.7}$$

and a backward model as

$$P(s^i) = \prod_{j=1}^{\text{length}(s^i)} P(s^i_j | s^i_{j+1:\text{length}(s^i)}) \tag{5.8}$$

where s^i_j is the j^{th} word in sentence i, and $\text{length}(s^i)$ is the number of words in sentence i including one `<eos>` symbol. For a forward model, the first word in a sentence is predicted given the `<eos>` symbol. All words in the sentence are used to predict the `<eos>` symbol at the end of that sentence. For a backward model, to prevent using words from other sentences,

instead of predicting the <eos> at the end of the sentence, the <eos> at the beginning of the sentence is predicted. The <eos> at the end of the sentence is used to predict the last word of that sentence.

Both forward and backward models give valid sentence probabilities of the same form. Therefore they can be combined together to get a valid log joint probability. The derivation of finding the weights for combining sentences is as follows:

$$\begin{aligned} \log P(w_{1:N}) &= \sum_{i=1}^{Z} \log P(s^i) \\ &= \sum_{i=1}^{Z} \log\{\sum_{j=1}^{K} \lambda_j P(s^i; m_j)\} \\ \implies \lambda_j^{(t+1)} &= \frac{1}{Z} \sum_{i=1}^{Z} \frac{\lambda_j^{(t)} P(s^i; m_j)}{\sum_{\ell=1}^{K} \lambda_\ell^{(t)} P(s^i; m_\ell)} \end{aligned} \tag{5.9}$$

To avoid including any context from other sentences when predicting the words in the current sentence, a sentence boundary reset is done when training the NNLMs. Different NNLMs have different ways to do sentence boundary reset.

Sentence Boundary Reset for FFLM

For a n-gram FFLM, the context is the n-1 words immediately before the target word. In order to prevent the context from containing words in previous sentences for forward model, or containing words from future sentences for backward model, an algorithm can be constructed to search for the index of the <eos> symbol that is closest to the target word in each context. Then every word from the beginning of the context to that idx is converted to <eos>. For example, if the context is [<eos>, w_1, <eos>, w_2] and the target word is w_3, the closest <eos> symbol to w_3 is the one between w_1 and w_2. After the sentence boundary reset operation, this context is converted to [<eos>, <eos>, <eos>, w_2], so that the prediction of w_3 is based only on the information from the current sentence.

Sentence Boundary Reset for RNNLM

At each time step, the embedding of the current word is concatenated with the hidden vector from the last time step, which carries the history information. For each stream, if the current word is <eos>, the sentence reset can be done by setting the hidden vector to be a vector of zeros. Then the information from the previous sentences no longer exists.

Sentence Boundary Reset for TLM

Two possible ways can be used to do sentence boundary reset for TLM. The first method is similar to sentence reset for FFLM. When using the training method shown in Figure 5.3, the goal is to predict a word rather than to predict a sequence. If the input is $w_{t-n:t}$, the target is w_{t+1}. The closest `<eos>` symbol can be found, e.g. w_i, then $w_{t-n:i}$ are set to `<eos>`.

As discussed before, the training is much slower if the target is a word rather than a sequence. The method currently used for training decoder style Transformer is able to give prediction for n words if there are n words in the input. Therefore, to preserve the ability for giving a sequence of predictions, the sentence reset needs to be done by modifying the attention mask. When predicting target words, the attention mask sets the attention scores between words which are not in the same sentence to zero. If a input sequence is [w_1, w_2, `<eos>`, w_3] and the target sequence is [w_2, `<eos>`, w_3, w_4]. Without a sentence boundary reset, the attention mask is a lower triangular matrix that masks future words (Figure 3.7). The mask that can do sentence reset is the following:

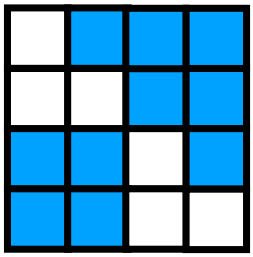


Fig. 5.9 `right mask` with sentence reset, blue positions have attention score zero

The mask for target word w_3 is the third row of the matrix in Figure 5.9. Only one element is white means that w_3 is predicted given only <eos>. Likewise, the fourth row of the matrix corresponding to w_4 shows that after sentence boundary resetting, the attention score connecting w_4 and w_1 and w_2 is zero.

PPL after Sentence Boundary Resetting

Using the methods mentioned above, the PPLs after sentence boundary resetting are shown in Table 5.9.

Compared to Table 5.6, the RNNLM and Transformer have higher PPL in all three sets after using the sentence boundary reset. This shows that these models can gather information from other sentences better than models like 5-gram FFLM. Since the models with sentence boundary reset assume that different sentences are independent, the sentence probabilities

Model	Direction	Train PPL	Dev PPL	Eval PPL
FFLM	=>	47.39	78.92	71.40
FFLM	<=	49.38	79.48	71.88
RNNLM	=>	33.15	68.23	61.24
RNNLM	<=	35.29	68.60	60.78
TLM	=>	41.36	71.66	63.52
TLM	<=	44.65	71.68	63.70

Table 5.9 NNLMs in Table 5.6 with sentence boundary reset

from forward and backward models can be combined and then a valid combined PPL can be obtained.

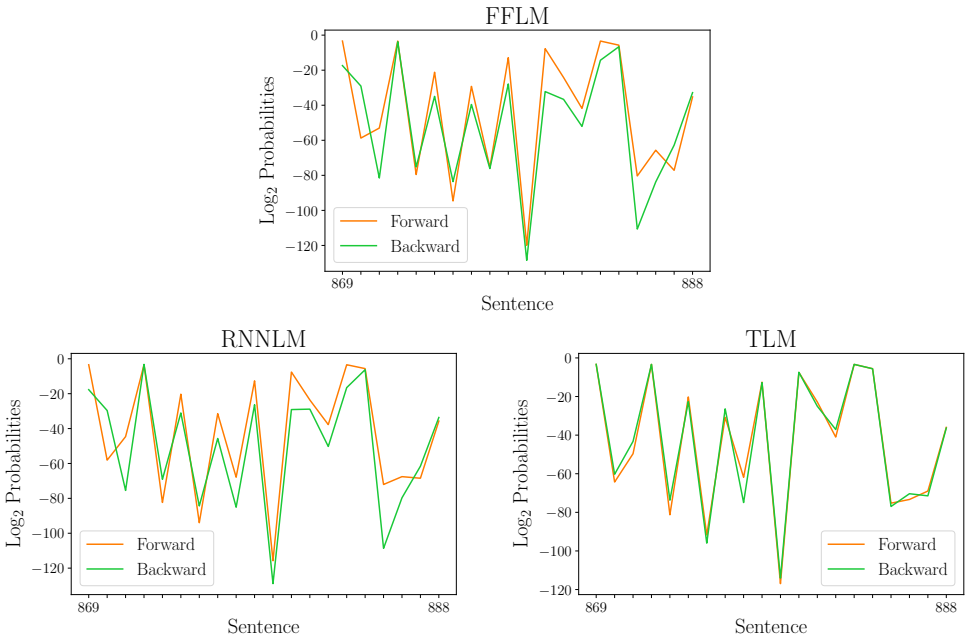


Fig. 5.10 Sentence probabilities from forward and backward models of the same type, all with sentence reset. 20 consecutive sentences are selected from the dev set

Since the sentences in forward models start from the first word to the <eos> symbol at the end, and in backward models a sentence start from the last word to the <eos> symbol at the beginning, it is worth checking if the sentence probabilities of forward and backward models align together. Figure 5.10 shows that the forward and backward models of the same type yield very similar trend, especially for TLM.

Combining Forward and Backward NNLMs of the Same Type

After combining sentence probabilities instead of word probabilities, the PPLs are higher for all three models compared to the non-valid PPLs in Table 5.8. Table 5.10 shows that forward model for all three types of NNLMs are more important than backward models.

Model	Dev PPL	Eval PPL	Forward Weight	Backward Weight
FFLM-C	44.37	42.59	0.58	0.42
RNNLM-C	38.12	36.14	0.58	0.42
TLM-C	67.86	60.39	0.51	0.49

Table 5.10 PPL after combining sentence probabilities from forward and backward models of the same type in Table 5.9. The suffix '-C' indicates the combination of the forward and backward models of the same type. Forward weight indicates the interpolation weight for forward model and backward weight is the weight for backward model.

After combining the sentence probabilities of forward and backward NNLMs of the same type, the next combination is to combine the sentence probabilities across different types of NNLMs. Table 5.11 shows that EM algorithm decides to put most of the weights on RNNLM-C (77%) and the least weight on FFLM-C (4%). These weights can be used in n-best LM rescoring task to interpolate the scores from different LMs.

Weights	FFLM-C	0.04
	RNNLM-C	0.77
	TLM-C	0.19
Combined PPL	Dev	37.62
	Eval	35.66

Table 5.11 PPL of the combination of all three models in Table 5.10

5.5.3 LM Rescoring Experiments for Speech Recognition

Two 50-best lists are generated for dev and eval set for computing WER. There are 13088 utterances in dev set and 12640 utterances in eval set. Each utterance has at most 50 hypotheses extracted from the lattice that was generated by an acoustic model and a tri-gram language model trained on AMI and Fisher. For each hypothesis the acoustic score is fixed while the language model score is replaced. The new language model score is either generated by a single NNLM or a combined NNLM. The final score of each utterance is computed by interpolating the original acoustic score and the new language model score.

Interpolating NNLMs with Sentence Boundary Reset using Weights that Minimise Dev PPL

The first approach to obtain word error rate is by the following steps:

1. For each utterance, read the hypotheses and save acoustic scores with the corresponding hypotheses.
2. Each available NNLM gives a score to each hypothesis, the score is the sum of the log probabilities of all words in the hypothesis including an added `<eos>` symbol at the end of the hypothesis for forward model. For backward model, an `<eos>` symbol is added to the beginning of each hypothesis and then the hypothesis is reversed before sending to the model.
3. If more than one NNLM are considered, the scores from all NNLMs are combined with linear interpolation to give the final LM score for each hypothesis. Weights are found by minimising the combined dev PPL in the previous section.
4. The acoustic score and the final LM score is linear interpolate, and LM scale γ in Equation 3.13 is manually tuned on dev set.

Model	Direction	Dev WER	Eval WER	LM scale
tri-gram-s	=>	27.8	28.8	15.0
tri-gram-b	=>	24.1	25.2	-
FFLM	=>	23.3	24.2	13.5
FFLM	<=	23.4	24.2	13.5
RNNLM	=>	22.4	23.3	14.5
RNNLM	<=	22.5	23.2	14.5
TLM	=>	22.6	23.4	14.5
TLM	<=	22.6	23.5	14.5
Oracle	-	13.1	13.4	-

Table 5.12 WER result for all NNLMs in Table 5.9 and the tri-gram models, after rescoring 13088 utterances in dev set and 12640 utterances in eval set. Tri-gram-s is trained only on AMI corpus while tr-gram-b is trained on AMI and Fisher. Oracle is obtained by using HResults rather than sclite, which does not include pre-processing for modifying hesitations and filtering synonym *etc*. Therefore, the oracle results are not fully comparable with other numbers, and possibly can have higher WERs.

If 1-best output is extracted from the original lattice, the results is the dev WER and eval WER for tri-gram-b in Table 5.12. The tri-gram-s trained on AMI corpus performs

worse than tri-gram-b trained on both AMI and Fisher corpora. The difference between the performance is limited since the comparison is implemented on the same 50-best list. Assuming the best choices over the the 50-best list for all utterances are selected, the oracle results is shown in the last row of Table 5.12, which can be used as a lower bounds for the WERs after LM rescoring. Similar to PPL, the forward and backward NNLMs of the same type also achieve very similar results in WER. The models in Table 5.12 are first combined with another model of the same type but with different direction. Then the resulting models are combined together with the weights in Table 5.11.

Model	Dev WER	Eval WER	LM scale
Baseline	24.1	25.2	-
RNNLM (=>)	22.4	23.3	14.5
FFLM-C	23.1	24.0	13.5
RNNLM-C	22.2	22.9	14.5
TLM-C	22.2	23.0	14.5
All NNLM-C	22.1	22.8	14.0

Table 5.13 Combine models of the same type with the weights in Table 5.10 and Table 5.11. All NNLM-C is the combination of FFLM-C, RNNLM-C and TLM-C. Baseline is the 1-best result scored by tri-gram-b.

As shown in Table 5.13, the combination of forward and backward NNLMs of the same type achieves lower WER than any single model that belongs to the corresponding type. The reduction varies from 0.2% to 0.4% in WER. A further 0.1% absolute reduction in both dev WER and eval WER is achieved by combining across different types of NNLMs, comparing to the best NNLM before combining across different types (RNNLM-C).

Interpolation Weights using CMA-ES

Having realised that the weights for each NNLM found to minimise dev PPL may not be the weights that also minimise WER, a second approach to interpolate LMs is considered. The Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen and Ostermeier, 2001) has the ability to determine interpolation weights such that the a black-box objective function is minimised. Instead of using weights that minimise dev PPL, CMA-ES can be used to serve as an optimisation algorithm which finds the weights to interpolate language model scores such that the dev WER is minimised. This method samples a sets of weights from a normal distribution in each iteration and updates the mean and covariance function based on the WER results obtained.

Weights	FFLM (=>)	0.12
	FFLM (<=)	0.01
	RNNLM (=>)	0.21
	RNNLM (<=)	0.27
	TLM (=>)	0.23
	TLM (<=)	0.07
WER	Dev	22.0
	Eval	22.8

Table 5.14 Weights found for each NNLM by CMA-ES and the final WER result, the LM scale is 16.1.

Comparing to the weights found by minimising dev PPL, the distribution of weights that directly minimises dev WER is less sharp. Although the weights found by CMA-ES achieve better result in dev WER, the result for eval WER does not show improvement. There is an increase of 0.1% in absolute value for eval insertion error using weights from CMA-ES. The conclusion is that both methods for finding interpolation weights for different LMs achieve similar performance. Using EM algorithm to find weights is much more efficient, if the final log probability is valid.

Weights	FFLM (=>)	0.11
	FFLM (<=)	0.04
	RNNLM (=>)	0.38
	RNNLM (<=)	0.04
	TLM (=>)	0.35
	TLM (<=)	0.07
WER	Dev	21.9
	Eval	22.7

Table 5.15 Allowing forward NNLMs in Table 5.14 to include history from previous sentences. Interpolation weights are found by CMA-ES.

However, since the PPL is not needed when using CMA-ES to find the weights for each NNLM, all forward NNLMs can now use the 1-best history, which are the past sentences with the highest rank after speech recognition LM rescoring. Some changes to the rescoring pipeline is needed. The 1-best history generated by each NNLM is saved and is available for the corresponding NNLM when rescoring the next sentence. The reason for not allowing backward NNLMs to see future sentences is that such procedure cannot be applied in real

time in an ASR system. After allowing the forward models in Table 5.14 to see information in previous sentences, the WER is shown in Table 5.15.

A further 1% absolute reduction in WER is achieved in both Dev WER and Eval WER after allowing forward NNLMs to see the future history. The result has shown that removing sentence boundary reset for forward NNLMs can improve the rescoring performance. No further improvement is observed by including tri-gram-b into the combination.

5.6 Fine-Tuning Pre-trained Models

Two pre-trained models are considered in experiments, which are GPT and BERT. One output layer is added on the top of these two models to transform the pre-trained hidden states to the distribution of the output dictionary.

5.6.1 GPT

Since GPT also uses decoder structure, the same training procedure and data arrangement for the 8-layer Transformer model shown before can be used in the fine-tuning process (the fine-tuned GPT language model will be called GPT-LM). However, the input should no longer be a word sequence. It needs to be converted into subword sequence by the same BPE algorithm. And the input dictionary is the same as the original GPT model, while the output dictionary is the tokenised version of the word dictionary from the AMI corpus.

The pre-trained model is trained without any special symbol to indicate where the sentence boundaries are. Therefore, instead of adding `<eos>` to the sentence boundary, a period is added to the end of each sentence and also the beginning of the dataset. This is because period should be the most common sentence boundary symbol in normal texts.

To fine-tune these models, the final linear layer will be removed from the pre-trained model and then a fully connected layer will be added, which projects the hidden states to in-domain (AMI) vocabulary. The hidden states generated by the GPT model are considered to be of high quality and there is a massive number of parameters waiting to be updated when fine-tuning all layers. Therefore only the added output layer will be trained on the AMI corpus during the fine-tuning process. It may be difficult to do sentence reset since the original GPT model is trained with history across sentence boundaries, which makes the sentence boundary reset challenging in fine-tuning stage. Two possible ways can be considered. The first one is to add attention mask in each of the 12 Transformer layers in the pre-trained GPT model, and fine-tune all layers during fine-tuning. The second one is to predict one target word at each time step so that words from other sentences can be change to

<eos> before sending to the model. However, these two methods largely increase the amount of time needed for fine-tuning the GPT model. Therefore, no sentence boundary reset is implemented with GPT pre-trained model. Although there are multiple meetings in the AMI corpus, GPT-LM can use the attention mechanism to decide whether to rely on the 1-best history or not.

The fine-tuned GPT-LM gives sub-word probabilities rather than word probabilities. To ensure that the PPL is comparable to the models in the previous section, the PPL is computed by:

$$\begin{aligned} \text{PPL} &= P(w_{1:\hat{N}})^{-\frac{1}{N}} \\ &= 2^{-\frac{1}{N}\sum_{i=1}^{\hat{N}} \log_2 P(\hat{w}_i|\hat{w}_{1:i-1})} \end{aligned} \tag{5.10}$$

where $\hat{w}_i$ indicates the i^{th} sub-word, $\hat{N}$ is the total number of sub-words after tokenising N words from the original dataset. The PPL and WER are shown in Table 5.16.

Model	Dev PPL	Eval PPL	Dev WER	Eval WER
Baseline	68	61	22.4	23.3
GPT-LM	56	50	21.6	22.4

Table 5.16 GPT LM after fine-tuning the last output layer; Baseline is the best single model (forward RNNLM). LM scale is set to 15.0 for fine-tuned GPT-LM

This result is better than every NNLM without combination shown before, and also better than any combination of NNLMs trained only on AMI corpus (0.3% absolute reduction in WER in both dev and eval WER comparing to Table 5.15). the which proves that the hidden states from pre-trained models are useful in the current task. The GPT model achieves 23.0% WER in dev set even without fine-tuning. After interpolating the fine-tuned GPT with FFLM-C, RNNLM-C and Transformer-C, further reduction in WER can be achieved, where the results are shown in Table 5.17.

Model	Dev WER	Eval WER
GPT-LM + FFLM-C + RNNLM-C + TLM-C	21.1	21.9

Table 5.17 GPT + FFLM-C + RNNLM-C + Transformer-C; The weights are 0.55, 0.02, 0.25 and 0.18; LM scale factor is 14.8, both weights and LM scale are found by CMA-ES

A further 0.5% absolute reduction in dev WER and 0.6% in eval WER are achieved. Even when interpolating with such a powerful model like GPT-LM which was pre-trained on much larger datasets, the LMs that are the interpolations of the forward and backward

models can still contribute to further WER reduction. This result emphasises that different NNLMs are highly complementary and a combination of NNLMs improves the performance significantly. Further combinations also include forward NNLMs train on AMI corpus that do not do sentence boundary reset, but no improvement is found in combined dev WER compared to Table 5.17.

5.6.2 BERT

BERT is an encoder style Transformer, which means that the model is able to see future information. The output of the softmax layer is no longer a conditional probability given previous words. For a bi-directional model, the output probability becomes:

$$P(w_i|w_{1:i-1}, w_{i+1:N}) \tag{5.11}$$

However, the model that can see bi-directional context may no longer be able to calculate a valid PPL. This is because the PPL requires the $P(w_{1:N})$, which can be decomposed by a uni-directional model as follows:

$$P(w_{1:N}) = \prod_{i=1}^{N} P(w_i|w_{1:i-1}) \tag{5.12}$$

which is the product of all softmax outputs from a uni-directional. The probability $P(w_i|w_{1:i-1})$ cannot be simply replaced by the output from a bi-directional model. With bi-directional model, $P(w_{1:N})$ may be computed by:

$$P(w_{1:N}) = (\prod_{i=1}^{N} P(w_i|w_{1:i-1}, w_{i+1})P(w_{1:i-1}, w_{i+1:N}))^{\frac{1}{N}} \tag{5.13}$$

Since for each i, $P(w_i|w_{1:i-1}, w_{i+1})P(w_{1:i-1}, w_{i+1:N}) = P(w_{1:N})$. Another way to compute the joint probability is by using the sum rule:

$$P(w_{1:N}) = \prod_{i=1}^{N} (\sum_{w_{i+1:N}} P(w_i|w_{1:i-1}, w_{i+1})P(w_{i+1:N}|w_{i+1:N})) \tag{5.14}$$

Since

$$\begin{aligned} \sum_{w_{i+1:N}} P(w_i|w_{1:i-1}, w_{i+1})P(w_{i+1:N}|w_{i+1:N}) &= \sum_{w_{i+1:N}} P(w_i, w_{i+1:N}|w_{1:i-1}) \\ &= P(w_i|w_{1:i-1}) \end{aligned} \tag{5.15}$$

Neither of the two methods is straightforward. They are all computationally intensive and optimisation is required. Previous work (Chen et al., 2017b) shows that by multiplying a tunable constant α to the hidden states before entering softmax layer, the bi-directional RNNLM out-performs uni-directional RNNLM in AMI dataset in terms of WER. The paper calls the PPL from bi-directional model pseudo PPL and claims that α can smooth and flatten the distribution so that the bi-directional RNNLM is less affected by the errors made in ASR system. The paper also shows that the combination of a uni-directional RNNLM and bi-directional RNNLM achieves a better result than a single uni-directional model, regardless of using the smoothing constant or not. Therefore, the fine-tuned BERT LM (BERT-LM) will be combined with other uni-directional model to see if any reduction in WER can be achieved.

In the experiments, $\frac{1}{8}$ of the words in the training data are randomly selected without replacement for each epoch. Then 32 consecutive words before and after the selected words will be used as the context. The selected words are then replaced by the special symbol `[mask]` and the model uses the contexts to predict the masked words.

However, although the pseudo PPLs from the bi-directional model are very low (20.58 for dev and 17.8 for eval), the dev WER is 25.7% and no improvement is found by interpolation this BERT-LM with GPT-LM. One hypothesis is that BERT-LM can take past sentences and future sentences into consideration, but future sentence is not available during speech recognition LM rescoring. Therefore, the pre-trained BERT is fine-tuned again. Since the model only has one target word for a given sequence of inputs. Any word in the input sequence that belongs to a future sentence after the current sentence with the target word is now set to `<eos>`. This future sentence boundary resetting helps to make the training and rescoring consistent in terms of the input information.

After fine-tuning with future sentence boundary reset, there is a 1.3% absolute reduction in word error rate in dev WER in comparison to the the dev WER of BERT-LM without future sentence boundary reset. The dev WER is now 24.5% and eval WER is 25.3%.

Combining GPT-LM and other NNLMs with BERT-LM reduces an extra 1% in both dev WER and eval WER (shown in Table 5.18). This finding is also consistent with the findings in Chen et al. (2017b), which shows that although bidirectional model may get a higher WER then uni-directional models, they help to reduce the WER after combining with uni-directional models.

Comparing to the best result in Sun (2019), which is 22.0% in dev WER, the result in this project has achieved a further 1% absolute reduction in dev WER.

Weights	FFLM (=>)	0.01
	FFLM (<=)	0.01
	RNNLM (=>)	0.20
	RNNLM (<=)	0.09
	TLM (=>)	0.14
	TLM (<=)	0.09
	GPT-LM (=>)	0.39
	BERT-LM (<=>)	0.07
WER	Dev	21.0
	Eval	21.8

Table 5.18 Combination of all fine-tuned LMs and NNLMs that are trained on AMI corpus only. All forward NNLMs do not have sentence boundary reset. LM scale is 15.1.

5.7 Sensitivity of the Correctness of History

Currently GPT-LM, BERT-LM and all other NNLMs that do not do sentence boundary reset use 1-best history during LM rescoring. Since each NNLM generates its own 1-best history, the quality of the history information is not consistent. This section investigates how the quality of the 1-best history affects the LM rescoring performance on the 50-best list. Instead of using different history, all models will be provided with the same reference history when predicting the current sentence.

The change in WERs after using reference history for individual NNLM is shown in Table 5.19.

Model	Dev WER	Eval WER	LM Scale
FFLM	23.3 (23.3)	24.3 (24.3)	13.7
RNNLM	22.2 (22.2)	22.9 (22.9)	14.8
TLM	22.4 (22.5)	23.2 (23.3)	13.9
GPT-LM	21.4 (21.6)	22.3 (22.4)	15.0
BERT-LM	24.4 (24.5)	25.3 (25.3)	11.1

Table 5.19 WER using 1-best history (inside the parenthesis) and using reference history (outside the parenthesis). All models are forward NNLMs except the bidirectional BERT.

The results show that all three Transformer based NNLMs (forward TLM, GPT-LM and BERT-LM) show some reduction in dev WER after using reference history, which may be because these models have the ability to use longer history, therefore they are more sensitive to the quality of the history. The final combined result also shows a further 1% absolute reduction in WER after using reference history (shown in Table 5.20).

Weights	FFLM (=>)	0.01
	FFLM (<=)	0.01
	RNNLM (=>)	0.20
	RNNLM (<=)	0.09
	TLM (=>)	0.14
	TLM (<=)	0.09
	GPT-LM (=>)	0.39
	BERT (<=>)	0.07
WER	Dev	20.9 (21.0)
	Eval	21.7 (21.8)

Table 5.20 Combination of all NNLMs with history reference used for all NNLMs that can use history. LM scale is 15.1.

Chapter 6

Conclusion and Future Work

This thesis focuses on combining NNLMs to reduce the speech recognition WER by rescoring the 50-best list. Individual NNLMs are first investigated. Results show that the RNNLM and the TLM perform better than the FFLM in terms of PPL and WER. This shows that these two models can better consider the history information.

The combination of NNLMs starts from combining forward NNLMs. An absolute decrease of around four points (dev PPL reduces from 61.71 to 57.75 and eval PPL reduces from 55.36 to 51.99) is observed in both dev and eval PPL. When trying to combine both forward and backward models of the same type, sentence boundary reset is conducted for each model in order to preserve a valid PPL. By assuming different sentences are independent, the n-best rescoring process does not require the information from other sentences, which has the potential to rescore multiple utterances at the same time. Although the performances of the forward and backward NNLMs of the same type are very similar, the combination largely reduces both PPL and WER.

Different ways for finding the weights to interpolate language model have been considered. The first method is to use weights which minimises the dev PPL and the second method is to use weights which directly minimises dev WER. Experiments show that both methods yield similar results in terms of both dev and eval WER.

More experiments have been implemented to fine-tune pre-trained models and then fine-tuned LMs are combined with NNLMs trained only on AMI corpus. Both GPT-LM and BERT-LM do not have sentence boundary reset for past sentences, so only one utterance from each meeting can be rescored at a time. The best result achieved in this thesis is the interpolation of the fine-tuned GPT-LM, BERT-LM, forward NNLMs (without sentence boundary reset) and backward NNLMs (with sentence boundary reset). The best result without using reference history is 21.0 for dev WER and 21.8 for eval WER, which has more than 3% absolute reduction in WERs compared to the baseline tri-gram WERs.

Other methods to combine different LMs can be investigated in the future such as training different NNLMs together instead of training them separately. A simple approach would be concatenating the last hidden states from different NNLMs and sending it to a softmax layer.

Future experiments can investigate methods that can compute a valid PPL or ways that can directly compare bi-directional methods with uni-directional models other than using WER. Possible changes can be conducted to improve the performance of BERT, such as multiplying a constant α to the un-normalised output probabilities (Chen et al., 2017a,b). Since using the reference history further reduces WERs for some models, it may be because of the model over-fits to the true text, which may be inconsistent to the n-best shortlist. In that case error sampling technique may be considered such as scheduled sampling (Bengio et al., 2015), which replace the true label by a sample label.

The 50-best list is fairly small, even for a small corpus like AMI. Larger n-best lists can be considered. Moreover, since the FFLM is basically an n-gram model, lattice rescore can be applied easily on this NNLM. By using history clustering methods (Liu et al., 2014), the number of distinct states for the RNNLM during lattice rescoring can be dramatically reduced. Hence it is also possible to do lattice rescoring for the RNNLM efficiently. Although it may still be expensive to do exact lattice rescoring for the TLM that considers a long history, a better n-best list can be extracted from lattice rescored by the RNNLM, and do the n-best rescoring with the combined model in this thesis.

Due to time constrain, the experiments are only implemented on AMI corpus. Other datasets such as WikiText corpus (Merity et al., 2016) can also be used to train the NNLMs. Other speech corpora such as Switchboard (Godfrey et al., 1992) or LibriSpeech (Panayotov et al., 2015), which have much longer hours of recordings, should also be investigated. With more data, the TLMs can use much longer sequence length and the effect of layer normalisation and positional encoding can be verified. Other than the three types of NNLM investigated in this thesis, more architectures can be combined together and investigation of how they alter the WER can be conducted.

References

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bahl, L. R., Souza, P. V. D., Gopalakrishnan, P. S., Nahamoo, D., and Picheny, M. A. (1991). Context dependent modelling of phones in continuous speech using decision trees. In *In Proceedings DARPA Speech and Natural Language Processing Workshop*, pages 264–270.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

Carletta, J., Ashby, S., Bourban, S., Flynn, M., Guillemot, M., Hain, T., Kadlec, J., Karaiskos, V., Kraaij, W., Kronenthal, M., Lathoud, G., Lincoln, M., Lisowska, A., McCowan, I., Post, W., Reidsma, D., and Wellner, P. (2006). The AMI Meeting Corpus: A Pre-announcement. In Renals, S. and Bengio, S., editors, *Machine Learning for Multimodal Interaction*, volume 3869, pages 28–39. Springer Berlin Heidelberg, Berlin, Heidelberg.

Chen, S. F. and Goodman, J. (1998). An Empirical Study of Smoothing Techniques for Language Modeling. *Harvard Computer Science Group Technical Report TR-10-98*.

Chen, X., Liu, X., Ragni, A., Wang, Y., and Gales, M. J. (2017a). Future word contexts in neural network language models. In *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 97–103. IEEE.

Chen, X., Ragni, A., Liu, X., and Gales, M. (2017b). Investigating bidirectional recurrent neural network language models for speech recognition.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. (2019). Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. *arXiv:1901.02860 [cs, stat]*.

Davis, S. and Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(4):357–366.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Gales, M. and Young, S. (2007). The Application of Hidden Markov Models in Speech Recognition. *Foundations and Trends® in Signal Processing*, 1(3):195–304.

Godfrey, J. J., Holliman, E. C., and McDaniel, J. (1992). SWITCHBOARD: Telephone speech corpus for research and development. In *[Proceedings] ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 517–520. IEEE.

Good, I. J. (1953). The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3-4):237–264.

Goodman, J. T. (2001). A bit of progress in language modeling extended version. *Machine Learning and Applied Statistics Group Microsoft Research. Technical Report, MSR-TR-2001-72*.

Graff, D., Wu, Z., MacIntyre, R., and Liberman, M. (1997). The 1996 broadcast news speech and language-model corpus. In *Proceedings of the DARPA Workshop on Spoken Language Technology*, pages 11–14.

Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195.

Hermansky, H. (1990). Perceptual linear predictive (PLP) analysis of speech. *the Journal of the Acoustical Society of America*, 87(4):1738–1752.

Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T., and Kingsbury, B. (2012). Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine*, 29(6):82–97.

Irie, K., Zeyer, A., Schlüter, R., and Ney, H. (2019). Language Modeling with Deep Transformers. *arXiv:1905.04226 [cs]*.

Katz, S. M. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Trans. Acoustics, Speech, and Signal Processing*, 35:400–401.

Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Kneser, R. and Ney, H. (1995). Improved backing-off for M-gram language modeling. In *1995 International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 181–184 vol.1.

Kombrink, S., Mikolov, T., Karafiát, M., and Burget, L. (2011). Recurrent neural network based language modeling in meeting recognition. In *Twelfth Annual Conference of the International Speech Communication Association*.

Kreyssig, F. L., Zhang, C., and Woodland, P. C. (2018). Improved TDNNs using deep kernels and frequency dependent Grid-RNNs. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4864–4868. IEEE.

Liu, X., Wang, Y., Chen, X., Gales, M. J., and Woodland, P. C. (2014). Efficient lattice rescoring using recurrent neural network language models. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4908–4912. IEEE.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. (2016). Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient Estimation of Word Representations in Vector Space. *arXiv:1301.3781 [cs]*.

Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *INTERSPEECH*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Mikolov, T. and Zweig, G. (2012). Context dependent recurrent neural network language model. In *2012 IEEE Spoken Language Technology Workshop (SLT)*, pages 234–239, Miami, FL, USA. IEEE.

Ney, H., Essen, U., and Kneser, R. (1994). On structuring probabilistic dependences in stochastic language modelling. *Computer Speech & Language*, 8(1):1–38.

Panayotov, V., Chen, G., Povey, D., and Khudanpur, S. (2015). Librispeech: An ASR corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210. IEEE.

Peddinti, V., Povey, D., and Khudanpur, S. (2015). A time delay neural network architecture for efficient modeling of long temporal contexts. In *Sixteenth Annual Conference of the International Speech Communication Association*.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Povey, D. and Woodland, P. C. (2002). Minimum phone error and I-smoothing for improved discriminative training. In *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages I–105. IEEE.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/languageunsupervised/language understanding paper. pdf*.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8).

Saon, G., Sercu, T., Rennie, S., and Kuo, H.-K. J. (2016). The IBM 2016 English Conversational Telephone Speech Recognition System. In *Interspeech 2016*, pages 7–11.

Sennrich, R., Haddow, B., and Birch, A. (2016). Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Sun, G. (2019). *Cross-Utterance Language Models*. MEng thesis, University of Cambridge.

Sundermeyer, M., Schlüter, R., and Ney, H. (2012). LSTM neural networks for language modeling. In *Thirteenth Annual Conference of the International Speech Communication Association*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. J. (1995). Phoneme recognition using time-delay neural networks. *Backpropagation: Theory, Architectures and Applications*, pages 35–61.

Wen, T.-H., Heidel, A., Lee, H.-Y., Tsao, Y., and Lee, L.-S. (2013). Recurrent neural network based personalized language modeling by social network crowdsourcing. In *Proc. Interspeech*.

Werbos, P. J. (1990). Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

Woodland, P., Gales, M., Pye, D., and Young, S. (1997a). The development of the 1996 HTK broadcast news transcription system. In *DARPA Speech Recognition Workshop*, pages 73–78. Morgan Kaufmann Pub.

Woodland, P. C., Gales, M. J., Pye, D., and Young, S. J. (1997b). Broadcast news transcription using HTK. In *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 719–722. IEEE.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., and Le, Q. V. (2019). XLNet: Generalized Autoregressive Pretraining for Language Understanding. *arXiv:1906.08237 [cs]*.

Young, S. (1996). Large vocabulary continuous speech recognition: A review. *IEEE signal processing magazine*, 13(5):45–57.

Young, S., Evermann, G., Gales, M., Hain, T., Kershaw, D., Liu, X., Moore, G., Odell, J., Ollason, D., and Povey, D. (2015). The HTK book (for HTK version 3.5). University of Cambridge, 2015.

Young, S., Odell, J., and Woodland, P. (1994). Tree-Based State Tying for High Accuracy Modelling. In *HUMAN LANGUAGE TECHNOLOGY: Proceedings of a Workshop Held at Plainsboro, New Jersey, March 8-11, 1994*.

Young, S. J. and Woodland, P. C. (1994). State clustering in hidden Markov model-based continuous speech recognition. *Computer Speech & Language*, 8(4):369–383.

Yu, H., Tomokiyo, T., Wang, Z., and Waibel, A. H. (2000). New developments in automatic meeting transcription. In *INTERSPEECH*.

Zhang, C., Kreyssig, F. L., Li, Q., and Woodland, P. C. (2019). PyHTK: Python Library and ASR Pipelines for HTK. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6470–6474.

Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 19–27.