Cross-Utterance Language Models

Guangzhi Sun

Supervisor: Prof. Phil Woodland
Dr. Chao Zhang

Department of Engineering
University of Cambridge

This thesis is submitted for the degree of
Master of Engineering

Trinity College June 2019

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 thesis contains fewer than 12,000 words including appendices, bibliography, tables and equations and has fewer than 50 figures.

Guangzhi Sun
June 2019

Acknowledgements

First, I would like to thank my supervisor, Prof. Phil Woodland and Dr Chao Zhang, for their supportive instructions, guidance and patience throughout my master's project. Weekly meetings with Prof. Woodland have always been a great pleasure. I was deeply involved and filled with motivations and inspiring ideas especially when I was obstructed by some difficulties. Dr Chao has always been reachable, discussing ideas, answering questions and kindly offering effective suggestions. I would also like to thank Florian Kreyssig and Eric Li in our group who offered timely help in all aspects of my project and my work-life balance. Finally, I would like to thank my parents, my girlfriend, and my friends at Cambridge, who made every moment of my undergraduate life memorable and enjoyable.

Abstract

Language modelling is the task to estimate the probability of a given sequence and is widely used in many applications such as machine translation and speech recognition systems. For speech recognition systems, the language model (LM) provides the prior information which allows the maximum *a posteriori* decoding to be performed. Traditionally, n-gram LMs have been widely used for the last few decades as they are easy to implement and can be well generalised to unseen data. However, there are two major disadvantages for n-gram LMs which are the data sparsity problem and the n-th order Markov assumption.

Recently, recurrent neural network language models (RNNLMs) have become increasingly popular, as it mitigates both problems with n-gram LMs. The data sparsity issue is addressed by representing each word in a low-dimensional, continuous space, and the long-term dependency is improved via the recurrent connection between the hidden and the input layer which carries the information of previously seen words. With more efficient GPU training techniques and optimisation algorithms, RNNLMs can be trained on large scale dataset. Meanwhile, two of its variants, the GRU and LSTM, have also been broadly used to produce state-of-the-art results. Nevertheless, due to the limited dimension of the representation space and the error back-propagation training algorithm, it still suffers from a limited long-term dependency and is usually used at the single utterance level for speech recognition systems. Therefore, effective incorporation of the cross-utterance level information remains an open research area to explore.

To better exploit the use of the surrounding context, a cross-utterance LM using a two-level LSTM-based structure is proposed in this project. The first-level LSTM is used to produce utterance embeddings which are then used to extract the context representation via some extraction networks. The second-level LSTM adopts an RNNLM adaptation framework where the auxiliary feature is the context representation which is concatenated with the word embedding and the hidden state vector to form the input to the recurrent layer. To begin with, different context extraction networks have been used to extract useful information from a set of fixed utterance embeddings. Then, in order to mitigate the criterion mismatch between the first and second level LSTM of the cross-utterance LM, joint-training algorithms are proposed. Different data arrangements for context representation are also proposed under the joint-training framework. Last but not least, to better imitate the real context which contains acoustic errors during LM training, an acoustic error sampling technique is proposed.

The first contribution of this project is the two-level cross-utterance LM and a set of context extraction networks. Experimental results show that the incorporation of context information with fixed utterance embeddings achieves a relative improvement of 5% in perplexity and 0.2 on word error rate (WER) compared to the vanilla LSTM LM. The second contribution of this project is the joint-training algorithm of the two LM components. A set of training techniques are experimented to achieve better optimisation. Besides, three different data arrangements for context representation are proposed as well whose merits and drawbacks are analysed in the experiments. Results indicate that a 12% relative reduction in perplexity and 0.3 absolute WER value reduction compared to the vanilla LSTM model are achieved. The third contribution of this project is the sampling technique with real acoustic errors which narrows the gap between the LM training and its application in speech recognition. Finally, future work directions have been suggested at the end of this thesis.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Overview of Language Models

Language modelling is one of the most critical components in various artificial intelligent systems such as automatic speech recognition (ASR) systems, translation systems, and dialogue systems. In particular, modern ASR systems calculate the posterior distribution of a word sequence $W = \{w_1, w_2, ..., w_n\}$ given a sequence of acoustic observations $\mathbf{O} = \{\mathbf{o_1}, \mathbf{o_2}, ..., \mathbf{o_N}\}$ which can be expanded using Bayes' theorem as shown in Eq. 1.1.

$$\hat{W} = \underset{W}{\text{argmax}} P(W|\mathbf{O}) = \underset{W}{\text{argmax}} P(\mathbf{O}|W)P(W), \tag{1.1}$$

where $P(\mathbf{O}|W)$ is the likelihood of the observation which can be modelled using the acoustic model, and $P(W)$ is the prior distribution of the word sequence which is estimated using the language model. The acoustic model is usually trained on audio data where the speech and its word sequence labels are given, and the language model is trained on text data where a large number of word sequences are available. Combining the acoustic and the language model probabilities, the posterior probability of a word sequence can be calculated. The word sequence with the highest posterior probability is chosen as the recognition result.

The language model (LM) computes the probability of a specific word sequence which can be decomposed into word prediction probabilities. Traditionally, n-gram LMs which condition the probability of the current word on a fixed number of previous words, have been widely used over the decades. N-gram LMs can be constructed using counting-based methods which are usually very fast. Hence these LMs are often used in the first-pass decoding stage, and is suitable for lattice rescoring algorithm. However, n-grams suffer from both inefficient use of available data known as the data sparsity problem, and very limited context dependency as it only takes previous $n-1$ words into account.

More recently, recurrent neural network language model (RNNLM) is widely used to rescore the hypothesis provided by the acoustic models. This type of LMs represents the history for prediction using a fixed-length vector in a continuous space which will be updated recurrently and used for each prediction step. Two common forms of variations of the RNNLM, the gated recurrent units (GRU) and the long-short term memory (LSTM) structure are developed to provide better long-term dependencies. With efficient training algorithms, RNNLM outperforms the n-gram in both perplexities, a measurement of the LM on its own, and word error rates (WER), a measurement of the ASR systems.

1.2 Motivation and Proposed Approaches

Although the introduction of RNNLMs improved performance over n-gram LMs via the continuous-space representation, problems such as the error back-propagation algorithm and the fixed dimensionality of hidden states still limit its long-term dependency. The back-propagation algorithm still causes the gradient estimation to be vanishing or inaccurate after being propagated for many steps. Besides, as the data is processed in proceeding order, the model tends to memorise the nearer context much better than the further. Moreover, the fixed-length history representation inherently has limited expressiveness. Consequently, when combined with acoustic models, these language models are usually used at a level of a single utterance or a couple of utterances. On the other hand, the information across different utterances, such as topic coherency and phrase repetitions in conversations, is also useful for estimating the probability of the next-word. This information is either missed out by the RNNLM or globally captured as auxiliary input features to the RNN through word frequency based methods, such as topic and genre vectors in [6] and the adapted uni-gram cache model in [7]. Therefore, a flexible, versatile and effective representation of the context across surrounding utterances is needed, and is of particular interest for conversational transcription systems. **Throughout this thesis, an "utterance" refers to a "sentence" in a conversational transcription system**.

In order to capture local context information and to further improve the efficiency of context incorporation, a cross-utterance language model is proposed in this project. First, the information of each surrounding utterance is encapsulated into a fixed-length vector known as the utterance embedding. It is extracted using a standalone LM referred to as the first-level LM. A context representation vector is extracted from those utterance embeddings. This context vector is then used to adapt the main RNNLM which is referred to as the second-level LM, so that it helps the prediction of the next word. Incorporation of such vectors provides local context information which changes as the LM proceeds to the next utterance. Different utterance embeddings and context vector extraction networks are proposed. Meanwhile, it is also believed that the joint-training of the two-level LMs alleviates the criteria mismatch, hence different joint-training schemes are implemented. Furthermore, to mitigate the dependency on the true context data, and to incorporate the future context properly for ASR tasks, acoustic error sampling is introduced.

1.3 Outline of the Thesis

This thesis contains 6 chapters and is structured as follows.

- Chapter 2 provides the background knowledge of the language modelling task in detail, and introduces RNNLM, its variants, its training algorithms, its adaptation and its application in ASR systems.
- Chapter 3 focuses on the context vector extraction methods and the training algorithms with fixed utterance embeddings.
- Chapter 4 introduces the joint-training algorithm for the cross-utterance language models under 3 different data arrangements.
- Chapter 5 provides an error sampling technique during training.
- Chapter 6 concludes this thesis with future directions of the project.

Chapter 2

Language Modelling

This chapter contains an introduction to language modelling. It begins with the formulation of the language modelling task and how language models (LMs) are evaluated. Then two commonly used types of LMs will be described which are the n-gram language model and the neural network-based language model (NNLM). In particular, the feed-forward NNLM, the basic form of RNNLM and its variants, the gated recurrent unit (GRU) and the long-short term memory (LSTM) will be explained. Attentions will also be paid to the training algorithms of RNNLMs. Furthermore, the adaptation of LMs with auxiliary features and conditions will be explained, followed by the description of LM interpolation methods. Finally, the application of LMs in ASR tasks will be accounted.

2.1 Task Formulation

Language modelling is to compute the probability of occurrence for a given word sequence, $P(W)$. It is core to a wide range of speech and language tasks such as speech recognition, machine translation and dialogue systems. Especially for speech recognition, it provides the prior for the possible word sequences without observing acoustic features.

The joint probability of all the words in the sequence can be decomposed into cascading conditional probabilities using the chain rule as shown in Eq. 2.1.

$$P(W) = P(w_0, w_1, ..., w_N) = \prod_{i=1}^{N} P(w_i | w_{i-1}, ..., w_1, w_0), \tag{2.1}$$

where N is the total length of the word sequence, and w_0 is always the symbol representing the start of the sequence. Therefore, the task of computing the joint-probability can be transformed into computing the word prediction probability given other words that the model has seen in this sequence which is referred to as the history information. Different LMs have different ways to represent the history information, and the quality of the representation for the history information significantly affects the performance of LMs.

The performance of LMs are usually evaluated by the *perplexity*. Given a word sequence of length N, the perplexity (PPL) of the LM is shown in Eq. 2.2.

$$PPL = 2^{-\frac{1}{N}log_2(p(W))} = 2^{-\frac{1}{N}\sum_{i=1}^{N}log_2(P(w_i|w_{i-1},\ldots,w_1,w_0))}. \tag{2.2}$$

A language model with lower perplexity is considered to provide more accurate word prediction probabilities than the one with higher perplexity. Hence, PPL provides an important metric to evaluate the quality of a language model on its own.

2.2 N-gram Language Models

Eq. 2.1 requires a full-history representation for the joint probability to be calculated, however, the word prediction has much stronger dependencies in preceding words than in those in the further history. Therefore, n-gram LM adopts the Markov assumption to truncate the number of words in the history to $n-1$ words as shown in Eq. 2.3.

$$P(w_i|w_{i-1},,w_1) \approx P(w_i|w_{i-1},,w_{i-n+1}) \tag{2.3}$$

Thus, each conditional probability can be estimated by counting the number of occurrences of such n-word sequence normalised by the number of occurrences of such (n-1)-word history sequence in the training corpus. Maximum likelihood estimation is used to train the LM.

The data sparsity problem and the n-th order Markov assumption are considered the two major issues associated with n-gram LMs. Because the estimation is based on the count of occurrences of partial word sequences in the training data, we need to have sufficiently large counts to estimate the probability with a small variance. As the order n increases, the number of possible partial sequences increases exponentially, so the occurrences become more sparse, with some sequences not present in the training data assigned zero probability by the model. To mitigate this problem, various *smoothing* methods have been applied to the probability estimation such as Katz Smoothing [26] and Kneser-Ney Smoothing [27]. Smoothing of an n-gram LM usually takes the discounting and back-off procedure. In particular for Kneser-Ney smoothing, if an n-gram appears in the training data, it will be discounted by subtracting a value $C < 1$ known as absolute discounting. Otherwise, a smoothed maximum-likelihood estimation for a lower order n-gram will be used to back-off the prediction. More details can be found at [30]. Kneser-Ney Smoothing will be used throughout this project.

In addition to the inefficient use of the history information in the data sparsity problem, the n-th order Markov assumption by ignoring words prior to the (n-1)-th preceding also reflects its limitations in capturing long-term dependencies within the sequence. The data sparsity issue is mitigated by using the feed-forward NNLM, it is not until the introduction of the RNNLM has the Markov assumption limitation been effectively addressed.

2.3 Feed-forward Neural Network Language Models

A feed-forward neural network, also known as multi-layer perceptron, is introduced in the context of word-based language modelling. It retains the n-gram language model structure as only n-1 words are considered for prediction. The network architecture is shown in Fig. 2.1.

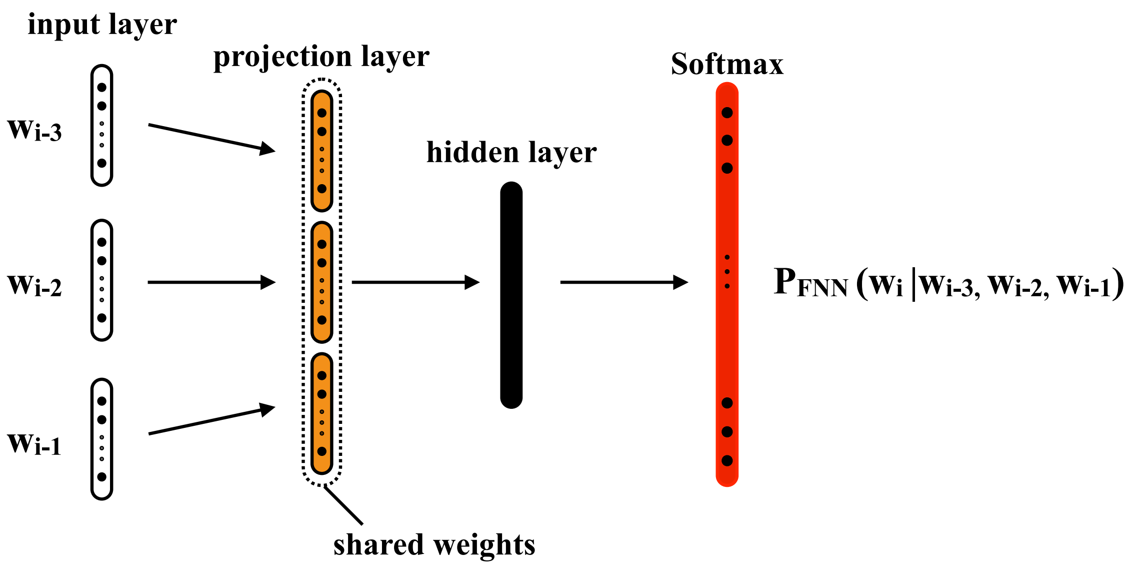


Fig. 2.1 A 4-gram Feed-forward neural network language model. The inputs are the previous 3 words represented by 1-of-K encoding, and are projected into a low-dimension vector with the shared projection layer. The output is a probability estimation of the current word.

Each input word is encoded as a 1-of-K vector where K is the number of word in the vocabulary (a list that contains all possible words), with 1 at the position of the word in the vocabulary and 0 otherwise. This vector is projected down to a trainable lower dimensional representation $\mathbf{x}_j$ with the relationship in Eq. 2.4, known as the word embedding.

$$\mathbf{x}_j = C\mathbf{w}_j, \tag{2.4}$$

where **C** is the shared weight matrix. The 3 word embeddings are then concatenated into a single vector **x** which is sent to the hidden layer shown in the following equation.

$$\mathbf{h} = f(W_x\mathbf{x} + \mathbf{b}), \tag{2.5}$$

where **h** is the output of the hidden layer, **b** and W_x are trainable parameters of this layer and f is the non-linear activation function. Finally, the output layer transforms the hidden layer output into a vector of dimension K, the same size as the input, and convert it into a valid probability distribution over the vocabulary V via the Softmax function shown in Eq. 2.6.

$$P_{FNN}(\mathbf{w}_i|\mathbf{w}_{i-1},...,\mathbf{w}_0) \approx \text{Softmax}(z_{\phi(\mathbf{w}_i)}) = \frac{\exp(z_{\phi(\mathbf{w}_i)})}{\sum_{\mathbf{w}\in V}\exp(z_{\phi(\mathbf{w})})}, \tag{2.6}$$

where $z_{\phi(\mathbf{w}_i)}$ is the i-th node of the final layer output. The parameters of the FNN including the projection matrix can be trained efficiently by the error back-propagation algorithm which will be described in detail in Sec. 2.4.2. One of the advantage of FNNLM is that the continuous-space representation of the words addresses the data-sparsity problem to some extent. These word embeddings allow the syntactic and semantic information of the words to be implicitly learned, which is not in the n-gram case. However, the FNNLM does not increase the dependency on a further history than a certain number of preceeding words.

2.4 Recurrent Neural Network Language Models

In order to address the dependency issue, RNNLM has been developed and widely used in recent years, and has demonstrated its ability to incorporate the information from the complete history in the sequence to predict the next word. RNNLM was first proposed in [1] where a single layer RNN was used whose superiority over a 5-gram LM with KN-smoothing was demonstrated. The work in [4] extended RNNLMs to use a Long-short term memory (LSTM) architecture that enriched the history representation. Meanwhile, it was discovered in [3] that further improvements could be achieved by linearly interpolating the RNNLM with n-gram LMs. Moreover, the work in [5] enabled efficient GPU training of RNNLMs with sentence bunches, and the toolkit in [8], in addition, supported faster training criteria such as noise contrastive estimation and variance regularisation. These methods allow RNNLMs to be trained on much larger corpora, and hence improve the performance evaluated by perplexity and also word error rate (WER) in ASR tasks.

The following sub-sections will first analyse the structure and functionality of the basic RNNLM. Then, the efficient training algorithm using spliced sentence bunches which exploits the use of the GPU will be described. Next, two variants of the RNN: GRU and LSTM structures will be introduced and their advantages will be discussed.

2.4.1 Basic RNNLM

The vanilla RNNLM model structure is illustrated in Fig. 2.2.

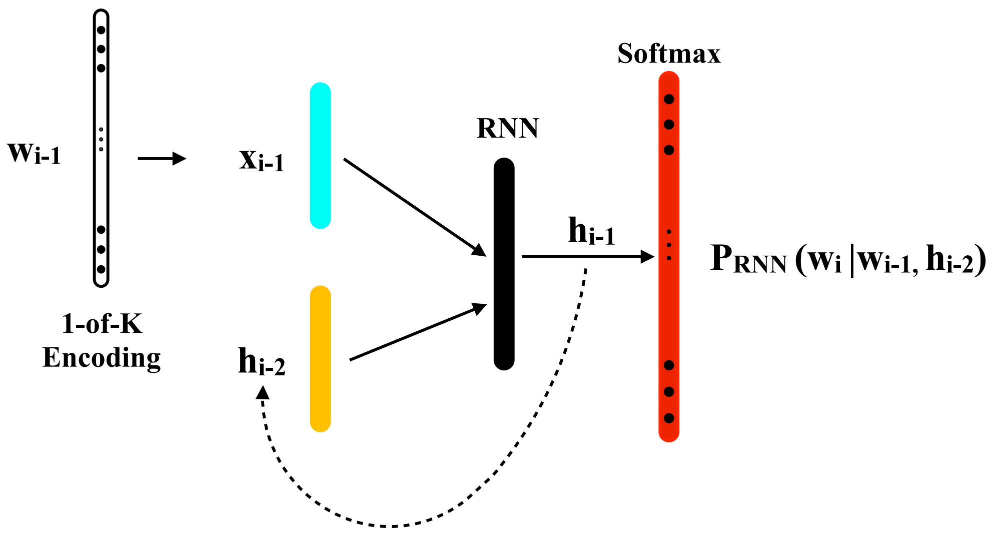


Fig. 2.2 Recurrent neural network language model which models the complete history with **h** in a continuous space. Projection layer is omitted.

Same as in the FNNLM, the 1-of-K encoding of each word $\mathbf{w}_i$ is converted into the word embedding before it is involved in the recurrent structure. RNNLM models the conditional probability distribution using the complete backward history which is represented with a continuous-space hidden state vector $\mathbf{h}_{t-1}$ of a fixed dimension. This history representation will be updated and used recurrently when every next word is being predicted, with the updating equation for the history representation (a.k.a. hidden state) shown in Eq. 2.7.

$$\mathbf{h}_{i-1} = f(W_x\mathbf{x}_{i-1} + W_h\mathbf{h}_{i-2} + \mathbf{b}_h), \tag{2.7}$$

which is simply a fully-connected layer with two weight matrices for the word embedding and the hidden state respectively, and with a non-linear activation function f. Thereafter, the updated hidden state $\mathbf{h}_{i-1}$ containing the information of $\mathbf{w}_{i-1}$ is carried forward to the next prediction step. The output layer is the same as FNNLM which produces an estimation of the probability distribution using the Softmax function.

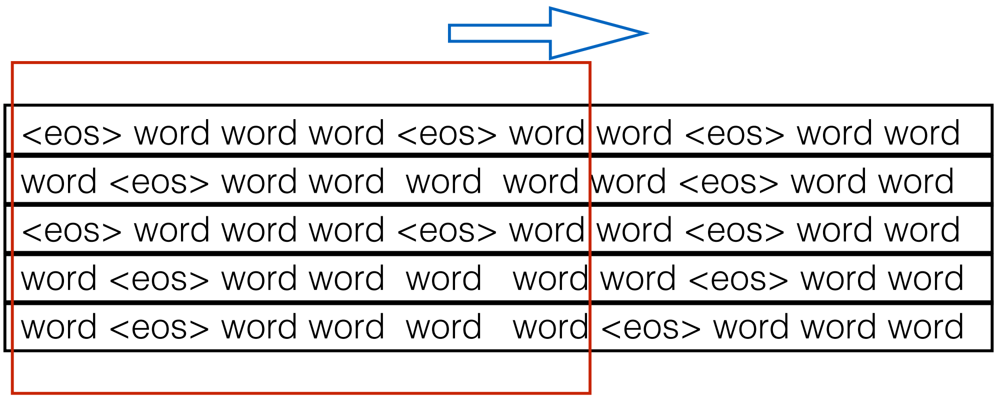
By using the history representation, the history for prediction is not limited to a fixed number of words anymore, but depends on the expressiveness of the hidden state vector. However, the fixed length of the history representation and the vanishing gradient problem caused by the training algorithm in Sec. 2.4.2 limit the ability of the RNNLM to memorise more than a few words. Therefore, variations of RNNLMs were developed to produce longer term dependencies which will be discussed in Sec. 2.4.3.

2.4.2 Training of RNNLMs

Because at each step the output of the network is a probability distribution over all words in the vocabulary, the cross-entropy between the predicted distribution and the target distribution which is 1 at the correct word and zeros otherwise, is used as the criterion for training. The form of the loss function is shown in Eq. 2.8.

$$L(\theta) = -\frac{1}{N}\sum_{j=1}^{N}\sum_{v\in V}\mathbb{I}_v(\mathbf{w}_j)\log(P(\mathbf{w}_j|\mathbf{w}_{j-1},\mathbf{h}_{j-2}), \tag{2.8}$$

where N is the size of the training set, and $\mathbb{I}$ is the indicator function. In order to use the first order optimisation methods, the gradients of the RNNLMs are calculated using an extended version of the error back-propagation algorithm, the back-propagation through time (BPTT) [32]. The essential idea of this approach is to treat the recurrent connection as a feed-forward one by unfolding the computational graph by a certain number of steps. As the use of activation functions such as sigmoid will produce a gradient less than 1, (e.g. less than 0.25 for sigmoid), and multiplying many times due to unfolding will cause the gradient value to decrease exponentially. This is known as the vanishing gradient problem. As the longer the gradient is being back propagated, the smaller the value is, hence the long-term dependencies are limited by the vanishing gradient problem. A vanished gradient is not only a waste of computational power, but will also cause inaccurate estimation due to finite precision. Therefore, when using the BPTT algorithm, it is a common practice to truncate to a fixed number of steps. The effect of BPTT steps will be further discussed in the experiments.



<eos> word word word <eos> word word <eos> word word
word <eos> word word word word word <eos> word word
<eos> word word word <eos> word word <eos> word word
word <eos> word word word word word <eos> word word
word <eos> word word word word <eos> word word word

Fig. 2.3 Parallel processing of 5 sentence streams for efficient GPU training of the RNNLM.

For the first-order optimisation to be performed on large data sets, mini-batch update needs to be used. In order to exploit the use of GPU parallel processing to achieve an efficient training scheme, the whole corpus is split into streams to be processed at the same time. The data arrangement is illustrated in Fig. 2.3.

The training method above is referred to as the spliced sentence bunch method [5]. The sentence boundaries are marked using `<eos>` which is treated as a word symbol and included in the vocabulary. As illustrated above, 5 streams of sentences are processed in parallel with a BPTT step of 6. For each mini-batch, a chunk in the red block (words at column 1 to 6) will be fed into the RNNLM to generate the prediction, and by using the target which is in the block shifted by one word to the right, gradients of the parameters are estimated. Then for the next mini-batch, words at column 7 to 12 will be processed.

2.4.3 Gated Recurrent Units and LSTM Language Models

In order to provide long-term dependencies to the RNNLM, the adoption of the gated recurrent units (GRU) [33] instead of vanilla RNN hidden layer alleviates the vanishing gradient problem by adding multiple gating connections to form a more complicated network component structure. A network gating is a vector, often output by a Sigmoid activation function, that acts as a probabilistic gate on the network values. It is a widely used method to provide direct connections. The commonly used GRU structure is illustrated in Fig. 2.4.

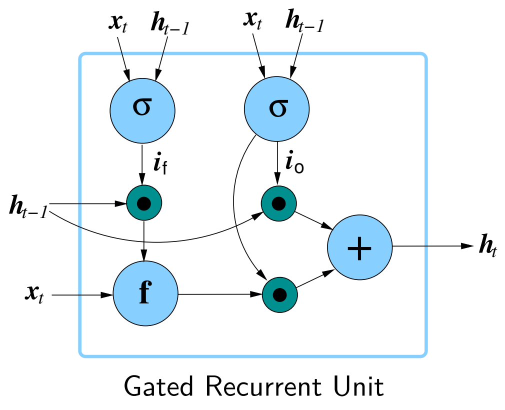


Fig. 2.4 Gated recurrent unit which contains a forget gate and an output gate that provide gating to both time and features. Figure taken from 4F10 lecture handout, university of Cambridge, engineering department.

In the figure above, $\mathbf{x}_t$ is the word embedding and $\mathbf{h}_t$ is the hidden state. $\mathbf{i}_o$ is the output gate and $\mathbf{i}_f$ is the forget gate. The forward path equations are shown below.

$$\mathbf{i}_f = \sigma(\mathbf{W}^f_f \mathbf{x}_t + \mathbf{W}^r_f \mathbf{h}_{t-1} + \mathbf{b}_f), \tag{2.9}$$

$$\mathbf{i}_o = \sigma(\mathbf{W}^f_o \mathbf{x}_t + \mathbf{W}^r_o \mathbf{h}_{t-1} + \mathbf{b}_o), \tag{2.10}$$

$$\tilde{\mathbf{h}}_t = f(\mathbf{W}^f_h \mathbf{x}_t + \mathbf{W}^r_h (\mathbf{i}_f \odot \mathbf{h}_{t-1}) + \mathbf{b}_h), \tag{2.11}$$

$$\mathbf{h}_t = \mathbf{i}_o \odot \mathbf{h}_{t-1} + (\mathbf{1} - \mathbf{i}_o) \odot \tilde{\mathbf{h}}_t, \tag{2.12}$$

where $\odot$ denotes the element-wise product. All the weights and biases can be updated during optimisation. The gating allows the model to preserve information through time by having the forget gate activation close to 1. Therefore, the gradients can be kept at a reasonable level during back-propagation, which empowers the model to learn longer-term dependencies.

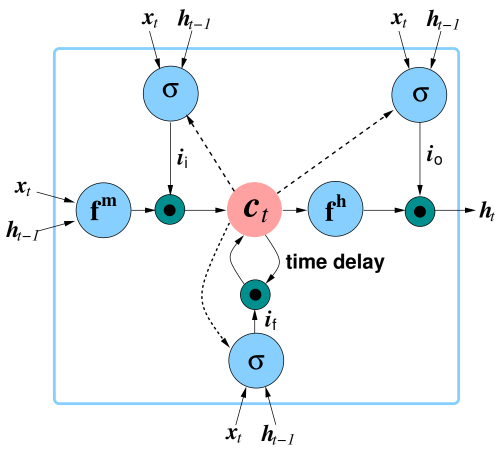


Fig. 2.5 LSTM structure with a memory cell for richer expression of the history. Figure taken from 4F10 lecture handout, university of Cambridge, engineering department.

In addition to adding gating to address the vanishing gradient problem, the long-short term memory (LSTM) structure [34] further enriches the expressiveness of the history by adding a memory cell. One of the commonly used LSTM structure is shown in Fig. 2.5. The memory cell is represented by $\mathbf{c}_t$ which will also be used recurrently as a part of the hidden states. The equations for the gating units are shown below.

$$\mathbf{i}_f = \sigma(\mathbf{W}_f^f \mathbf{x}_t + \mathbf{W}_f^r \mathbf{h}_{t-1} + \mathbf{W}_f^m \mathbf{c}_{t-1} + \mathbf{b}_f), \tag{2.13}$$

$$\mathbf{i}_i = \sigma(\mathbf{W}_i^f \mathbf{x}_t + \mathbf{W}_i^r \mathbf{h}_{t-1} + \mathbf{W}_i^m \mathbf{c}_{t-1} + \mathbf{b}_i), \tag{2.14}$$

$$\mathbf{i}_o = \sigma(\mathbf{W}_o^f \mathbf{x}_t + \mathbf{W}_o^r \mathbf{h}_{t-1} + \mathbf{W}_o^m \mathbf{c}_t + \mathbf{b}_o), \tag{2.15}$$

where $\mathbf{i}_i$ is the input gate with the same functionality as the other two gates. Then the updating equations for the hidden states are shown in Eq. 2.16 and Eq. 2.17.

$$\mathbf{c}_t = \mathbf{i}_f \odot \mathbf{c}_{t-1} + \mathbf{i}_i \odot f^m(\mathbf{W}_c^f \mathbf{x}_t + \mathbf{W}_c^r \mathbf{h}_{t-1}), \tag{2.16}$$

$$\mathbf{h}_t = \mathbf{i}_o \odot f^h(\mathbf{c}_t). \tag{2.17}$$

By inspecting the form of the memory cell which contains a gated connection to the memory cell of the last step, and another gated connection to the information of the current step, i.e. $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$, the longer dependencies are thus established via those gate connections. The equation for $\mathbf{h}_t$ which is used to produce the predictive probabilities is effectively fetching information that are useful for the current prediction from the memory cell via the output gate and a non-linear activation function.

Though LSTM structure contains much more parameters than the normal RNN structure, it performs better in many cases such as language modelling. With proper software implementation and training algorithms, it achieves a reasonable training speed on modern GPUs, and hence will be used as one of the building block for the cross-utterance language models.

2.5 Adaptation of RNNLMs

Normally the training corpus for a language model is collected from a variety of sources with different content, written or speaking form, genre or topic. The style of corpus and the context can heavily influence words to be used as well as the meaning of the word. When applied to unseen test data, mismatch arises in various aspects including speaking style and topic, which degrades performance as a result. Therefore, RNNLM adaptation is an important technique to mitigate the variation in genre, topic and speaker etc, and is very useful for practical application.

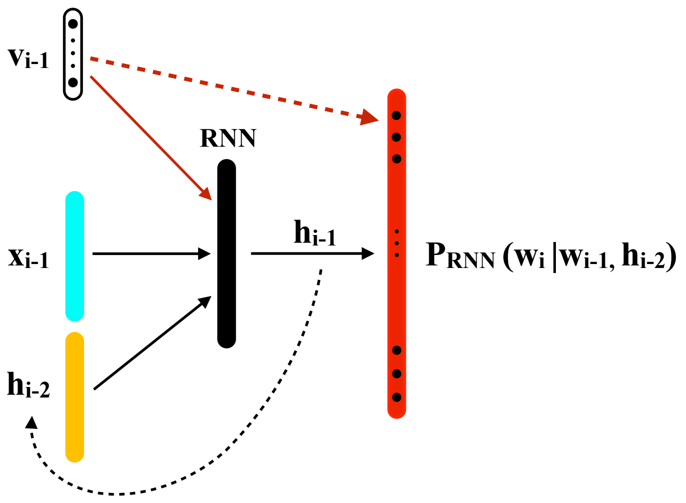


Fig. 2.6 RNNLM with additional information features as auxiliary inputs for LM adaptation.

One important method to perform RNNLM adaptation is to incorporate the informative features as an auxiliary input. These features are usually concatenated with the word embeddings to be fed into the hidden layer, and they will be fed into the output layer in some occasions as well [31]. Fig. 2.6 illustrates the incorporation of the auxiliary input, **v**.

A range of auxiliary features have been investigated for RNNLM adaptation in some earlier researches. For instance, morphological and lexical features in factored RNNLMs [15]; topic information derived from latent Dirichlet allocation (LDA) models in [16]; personalized user information such as demographic features exploited in [17]; utterance length information and lexical features were used in [18].

More recently, genre information such as talk show or sci-fi program has been incorporated for multi-genre broadcast data [6]. More sophisticated topic modelling has also be introduced to model specific documents, including probabilistic latent semantic analysis (PLSA)[35], the aforementioned LDA [36] and hierarchical Dirichlet processes (HDP) [37]. Furthermore, uni-gram and fully-connected context modeling has also been proposed which biases estimation probabilities and which various more locally with the change of context [7]. Furthermore, DNN-based embedding features representing the surrounding context is proposed which adapts the RNN layer as well as the bias of the Softmax layer [2].

2.6 Language Model Interpolation

Language model interpolation is widely used to combine multiple language models. Individual language model is trained on corpus from different domains, hence may have advantages in different aspects. These language models can be combined in test time. One commonly used combination method is the linear combination shown in Eq. 2.18.

$$P(\mathbf{w}|\mathbf{h}) = \sum_{k=1}^{K} \lambda_k P_k(\mathbf{w}|\mathbf{h}), \tag{2.18}$$

where K is the number of LMs to be combined and λ_k is the combination weight for system k. The set of combination weights can be obtained using the EM algorithm on a held-out validation set. This can be viewed as the update for a mixture model where the expectation step estimate the mixture weights, λ's and the maximisation step optimises the auxiliary function which provides a lower bound for the likelihood.

In this thesis, the interpolation is always between an RNNLM trained on in-domain data and an n-gram LM trained on much larger data corpus, the interpolation is shown below.

$$P(\mathbf{w}|\mathbf{h}) = \lambda P_{RNNLM}(\mathbf{w}|\mathbf{h}) + (1-\lambda)P_{ngram}(\mathbf{w}|\mathbf{h}). \tag{2.19}$$

Hence instead of using the complicated EM algorithm, λ could be tuned as a hyper-parameter on the held-out validation set manually to achieve the best performance on certain metric.

2.7 Application of Language Models in ASR

LMs are incorporated into speech recognition tasks via the rescoring process which combines the score for each sequence from the acoustic model with the score from the language model. The combination is performed by a weighted summation of the two scores which are both log-probabilities, and hence it reflects the posterior probability in Eq. 1.1. Normally the language model scale factor is tuned to find the best fit on the validation set. There are two ways to perform this combination: the lattice rescore and the n-best rescore algorithms.

A word lattice is a compact graph containing possible paths during recognition where each node in the graph is associated with time, word, acoustic score and language model score information. Lattice rescoring is to find the best path in the graph. As the history representation of the graph is complicated and hence RNNLM hidden state requires a lot of approximations, lattice rescoring is more often used for n-gram LMs. On the other hand, n-best rescore is more often used with RNNLMs. N-best rescoring re-ranks the shortlist of n most likely utterances based on the combination of language model score and acoustic model score for each utterance. The standard n-gram LMs are often used in the decoding to generate the lattices, and the n-best shortlist is extracted from the lattices. Then, the RNNLM, often linearly interpolated with an n-gram LM trained using much larger corpus, provides the LM score for the utterance which is the log-probability of the sequence, $P(W)$. The total score for re-ranking is shwon below.

$$\text{TotalScore} = \text{AcousticScore} + \gamma \times \text{LMScore}, \tag{2.20}$$

where γ is the hyper-parameter that can be tuned for each language model. The best utterances are then selected from the n-best list according to the total score to form the final hypothesis which will be scored by the Levenshtein distance.

Chapter 3

Cross-Utterance Language Models

In order to capture the context information and use it to achieve the adaptation of the RNNLM, the cross-utterance LM is proposed. The cross-utterance LM uses a two-level architecture where each level contains an individual RNNLM. Utterance embeddings are extracted using the first-level LM for all the surrounding utterances, and these embeddings are then used to extract the context representation. The second-level LM resembles the structure for LM adaptation where the context representation is used as an auxiliary input feature to the hidden layer only. In this chapter, the two-level model architecture will be described and analysed in detail, together with context vector extraction mechanisms. Training algorithms with fixed utterance embeddings will be provided. Thereafter, experimental setup which will be applied to all experiments throughout the thesis will be given, followed by results and discussions on the strength and weaknesses of different model architectures.

3.1 Two-Level Language Model Architecture

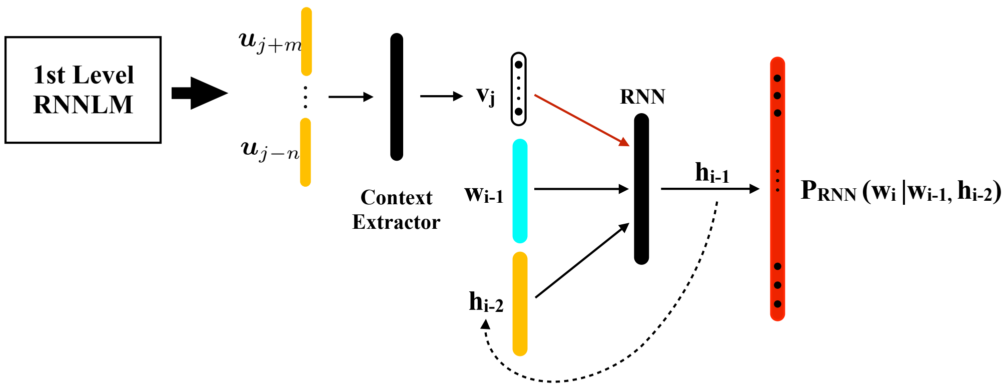


Fig. 3.1 The two-level cross-utterance language model where both levels are RNNLMs

The structure of the cross-utterance language model is shown in 3.1 where both levels employ the RNNLM with an LSTM structure. The first-level LM extracts the utterance embeddings $\mathbf{u}_{j+k}$ where the subscript j is the index of the current utterance and m, n are the index shift from the current utterance. In order to avoid knowing the true label before prediction, the current utterance embedding is excluded (i.e. $m,n = +1,+2,...$). Then these utterance embeddings are fed into a context extractor which produces the context vector $\mathbf{v}_j$. There are different network structures of the extractor such as feed-forward, recurrent or with attention mechanism, which will be discussed in the next section. The second-level RNNLM is then adapted to the context vector, and is trained using normal RNNLM training algorithms.

The context representation varies as it moves to the next utterance, but this variation is local and relatively smooth as most of the utterance embeddings still remain as the input to the extractor. Therefore, this architecture is able to capture continuously varying local context information. Besides, the neural networks use the local context data in a more efficient way than the unigram adaptation method. It also provides a more direct access to the surrounding utterances which ameliorates the degradation of gradient estimation for long back-propagation steps. Furthermore, the context representation enriches the expressiveness of the history as it increases the dimension of the representation space.

3.2 Context Vector Extraction Networks

Generally, the concatenation of all utterance embeddings is often too long to be efficiently used for language model adaptation, especially when it is trying to cover a wider range of context. Therefore, context extraction mechanism is adopted which integrates the utterance embeddings into a compact representation. The integration method here leverages the universal function approximator nature of neural networks to achieve a trainable and flexible transformation. The most straight forward way for such extractor is to use a fully-connected layer which directly maps the concatenated utterance embeddings to a lower dimensional representation space. Moreover, self-attention mechanism which provides dynamic combinations over all the utterance embeddings is also an alternative for the context extractor, similar to the one in [14]. In order to incorporate the sequential information, a hierarchical RNN can be applied by feeding the utterance embeddings to another RNN in their original order.

To begin with, the fully-connected layer context extraction takes the form in Eq. 3.1.

$$\mathbf{v}_j = f(W_v\mathbf{c}_j + \mathbf{b}_v), \tag{3.1}$$

where $\mathbf{c}_j^T = [\mathbf{u}_{j-n}^T, ..., \mathbf{u}_{j-1}^T, \mathbf{u}_{j+1}^T, ..., \mathbf{u}_{j+m}^T]$ is the concatenation of the utterance embeddings. The fully-connected layer is a fast, simple and effective way to extract the context vector.

Instead of the fixed transform layer, the multi-head self-attentive linear combination of the utterance embeddings is also adopted. As different utterances may have different influence to the context vector, which mainly depends on the content of these utterances, the self-attentive layer is introduced to achieve a input-dependent dynamic linear combination. The annotation matrix A, computed from the input vectors, provides the combination weights. Each column of the annotation matrix is an annotation vector which gives a set of scaling factors that weights the importance of each input before summing them. Specifically, if the input context covers a range of $m+n=T$ utterances, the utterance embeddings form a $T\times n$ matrix $C_j=[\mathbf{u}_{j-n},...,\mathbf{u}_{j-1},\mathbf{u}_{j+1},...,\mathbf{u}_{j+m}]^T$ where n is the dimension of each embedding, the annotation matrix can be calculated using Eq. (3.2) and applied to the inputs as in Eq. (3.3)

$$A_j = \text{Softmax}(\tanh(C_j W_1)W_2), \tag{3.2}$$

$$E_j = A_j^T C_j, \tag{3.3}$$

where $\mathbf{E}_j$ ($h\times n$) is the output and $\mathbf{A}_j$ ($T\times h$) is the h-head annotation matrix for utterance j. $\mathbf{A}_j$ is generated by passing the input matrix through two fully-connected layers with weight matrices $\mathbf{W_1}$ and $\mathbf{W_2}$ respectively, and the Softmax is performed column-wise to ensure each annotation vector sums to one. Moreover, when a multi-head self-attentive layer is used (i.e. $h>1$), to encourage different heads to extract dissimilar information, a penalty term in Eq. (3.4) is added to the cross-entropy loss function during training [13].

$$P=\mu||A_j^TA_j-I||_F^2=\mu\Big(\sum_i^h(\mathbf{a}_i^T\mathbf{a}_i-1)^2+\sum_{i,k,i\neq k}^h(\mathbf{a}_i^T\mathbf{a}_k)^2\Big), \tag{3.4}$$

where $\mathbf{a}_i$ are the annotation vectors for, I is the identity matrix and $||\cdot||_F$ denotes the Frobenius norm. The degree of influence of this term is adjusted by μ. As all terms in Eq. (3.4) are non-negative, minimising the cross terms, $(\mathbf{a}_i^T\mathbf{a}_k)^2$, encourages the annotation vectors to be orthogonal, while minimising the diagonal terms $(\mathbf{a}_i^T\mathbf{a}_k-1)^2$ encourages the annotation vectors to have fewer non-zero terms, ideally being one-hot vectors. Therefore, the effect of the penalty term will give high weights on different but few terms in the annotation vectors.

The multi-head output matrix E_j will then be reshaped into a vector, $\mathbf{E}_{j,reshape}=[\mathbf{e}_1^T,\mathbf{e}_2^T,...,\mathbf{e}_h^T]$, where $\mathbf{e}_i$ denotes each column of the output matrix. Then again transformed using a fully-connected layer to get a more compact representation as in 3.5.

$$\mathbf{v}_j = f(W_v\mathbf{E}_{j.reshape}+\mathbf{b}_v), \tag{3.5}$$

To further exploits the flexibility of the attention mechanism, the generation of the annotation vectors could also take the current word embedding into account. This is implemented by appending the current word embedding to each of the utterance embeddings to be combined.

After combination, the word embedding part will be removed from each output head and the rest will be transformed to the context vector. Furthermore, this network structure was originally proposed in [10] for utterance embeddings extraction, hence it will also be used for utterance embedding extraction methods in the later sections.

Unless positional information is explicitly added into the embeddings, two extraction methods mentioned above are not exploiting the sequential order of the utterances. Though it still remains disputable whether the positional information is better captured using attention or recurrent structure, hierarchical RNN structure has been proposed recently in many systems such as the document modelling system in [11] and in end-to-end ASR systems in [12]. The context extractor RNN is shown in Fig. 3.2 where two separate RNNs (LSTMs in this project) are used for the previous and the future utterances respectively.

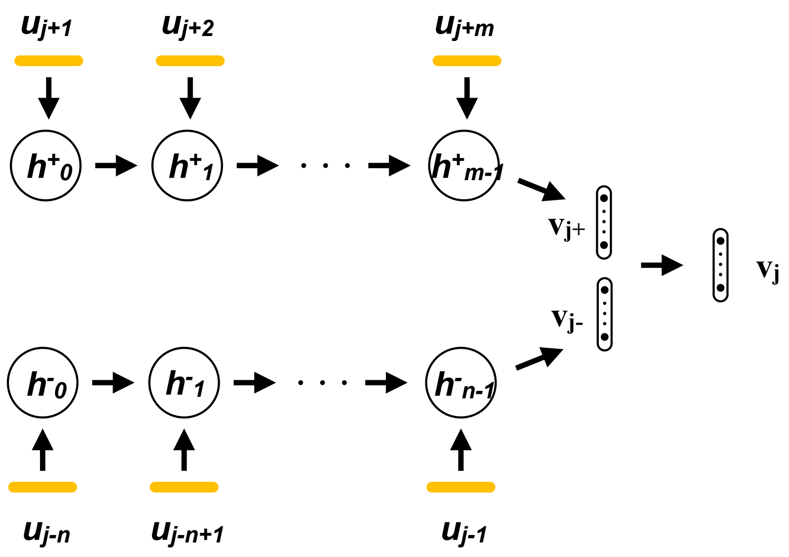


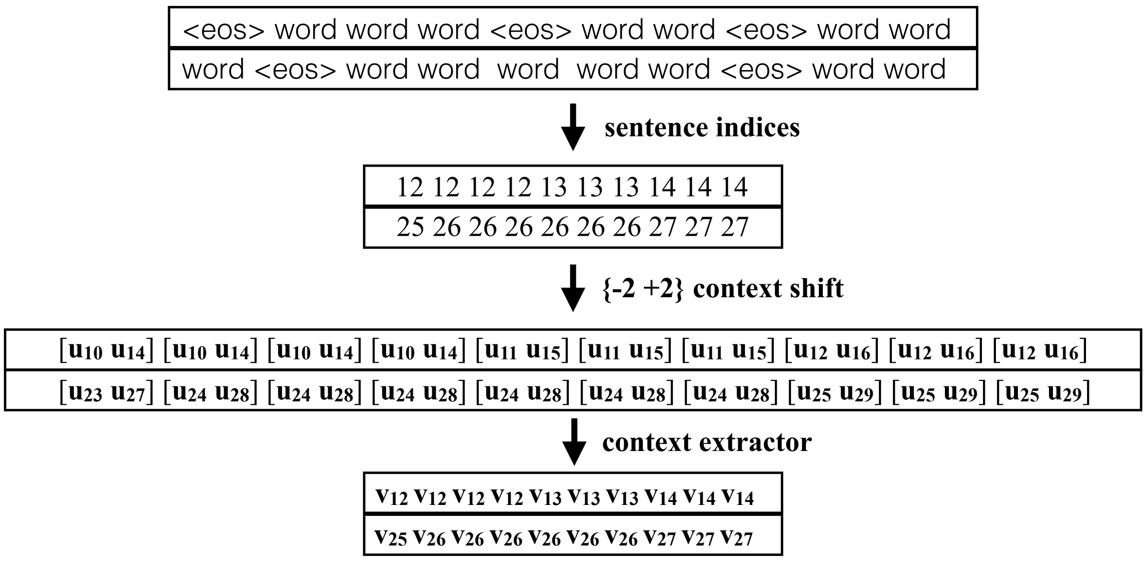
Fig. 3.2 Hierarchical RNN for context vector extraction

The sequential information could also be incorporated in the attention mechanism by using several extra bits representing the relative position. The three different extractor structures could also be combined. For example, self-attentive combination could also be used to combine the hierarchical RNN output at each step. Last but not least, the extraction networks will be jointly optimised with the second-level RNNLM.

3.3 Training with Fixed Utterance Embeddings

As joint-training of the two-level LM requires more complicated data arrangements, algorithms as well as optimisation strategies, the next chapter is dedicated to the description of the joint-training. As a starting point, training of this cross-utterance LM follows a two-stage pipeline where the first-level LM parameters which are trained in advance, are fixed while the second-level LM is being optimised.

In the first stage, a standalone LSTM-LM is trained on the text data until convergence. utterance embeddings are then obtained by forward propagating each utterance through this trained model, and store the memory cell at the end of each utterance since the memory cell provides richer information of the history as analysed in Sec. 2.4.3. In the second stage, the LSTM-LM is trained on the same text using the training data arrangement mentioned in 2.4.2. Besides, each word in the text is associated with an utterance index. Therefore in addition to the word index data chunk, a chunk of the same size is created which contains the index of the utterance each word belongs to. Next, the context associated to each word is obtained by applying the context shift which is a set configured manually (e.g. {-2, +2}). The utterance embeddings are then fetched and arranged according to the utterance indices, and will be sent to the context extractor. Fig. 3.3 illustrates this process. With this arrangement, the utterance embeddings can be easily processed under tensor operations, and it is also convenient to interact with the chunk of word embeddings.



<eos> word word word <eos> word word <eos> word word
word <eos> word word word word word <eos> word word
sentence indices
12 12 12 12 13 13 13 14 14 14
25 26 26 26 26 26 26 27 27 27
{-2 +2} context shift
[u10 u14] [u10 u14] [u10 u14] [u10 u14] [u11 u15] [u11 u15] [u11 u15] [u12 u16] [u12 u16] [u12 u16]
[u23 u27] [u24 u28] [u24 u28] [u24 u28] [u24 u28] [u24 u28] [u24 u28] [u25 u29] [u25 u29] [u25 u29]
context extractor
v12 v12 v12 v12 v13 v13 v13 v14 v14 v14
v25 v26 v26 v26 v26 v26 v26 v27 v27 v27

Fig. 3.3 Flowchart illustration of the batch arrangement for the training with fixed utterance embeddings. $\mathbf{u}_i$ are utterance embeddings extracted using the first-level LM

3.4 Experimental Setup

3.4.1 Data

Different language models are tested on penn-treebank (PTB) and the Augmented Multiparty Interaction (AMI) corpus. The AMI corpus is also used to test the performance for LM rescore.

PTB which is one of the standard dataset for LM test only, contains paragraphs taken from the Wall street journal. The vocabulary size of this dataset is 10000. The AMI dataset contains 100 hours of recorded meetings where there are typically 4 to 5 speakers in each meeting. The text source is the manual transcription from these meetings, which has a vocabulary size 13077. The sizes of AMI and PTB are shown in Table. 3.1 and Table. 3.2.

set name	train	dev	eval
number of words	911K	108K	102K
duration	80hrs	10hrs	10hrs

Table 3.1 Sizes and durations of each set in the AMI meeting data

set name	train	validation	test
number of words	930K	73K	82K

Table 3.2 Sizes of each set in the penn-treebank data

3.4.2 Models

N-gram LM, different RNNLMs and the cross-utterance LM will be compared in the following experiments, where the n-gram LM and the RNNLMs will be used as baselines. In particular, a counting based 4-gram LM is constructed using only the in-domain data with Kneser-Ney smoothing [19]. Different types of RNNLMs are implemented using PyTorch which is an open source python library designed for deep learning. Then, a single-layer vanilla LSTM LM is chosen as the baseline LM to compare with the cross-utterance LM. The first-level LM in the cross-utterance LM is configured in the same way as the standalone LSTM, with an equal or smaller size. The embeddings and hidden states of the second-level LM are of the same dimension as the standalone LSTM LM for fair comparison. All RNNLMs are trained using spliced sentence bunch scheme, with stochastic gradient descent (SGD) optimisation algorithm. Moreover, in order to be consistent between training and rescore which processes utterances separately, hidden states are reset to **0** (i.e. the all-zero vector) at the sentence boundaries which are marked with the "<eos>" symbol.

3.4.3 Rescoring Pipeline

In order to compare the performance in ASR tasks, n-best rescore is implemented. The acoustic model uses the ResNet-TDNN [20] with 4 ResNet blocks implemented in HTK 3.5.1 and PyHTK

[22]. The cross-entropy (CE) criterion and the minimum phone error (MPE) [21] criterion are used to train the acoustic model. After the word lattice generated, the lattice is rescored using a tri-gram to produce a 50-best list where n-best rescore is performed.

For vanilla LSTM LM, the LM score is obtained by forwarding each utterance in the list through the model and sum up the cross-entropy for each word in the sentence. Then, after summing up the acoustic model score and the LM score, utterances with the best combined score are written out to form a 1-best list. For the cross-utterance LM, the 1-best list generated using the vanilla LSTM is processed again to obtain the utterance embeddings based on the hypotheses, and then the second-level LM is run on the 50-best list with these embeddings. As before, utterances with highest combined score will be used to form the 1-best list.

3.4.4 Evaluation Metrics

The evaluation metric for LMs on their own is the *perplexity* (PPL) introduced in 2.1. Specifically for RNNLM, if the cross-entropy is calculated with base 2, then the PPL for the test set can be expressed as follows.

$$\mathrm{PPL} = 2^{-\frac{1}{N}\sum_{i=1}^{N} CE_i}, \tag{3.6}$$

where CE_i is the cross-entropy loss of the i-th word prediction. Hence it can be easily obtained by averaging the negative cross-entropy over the entire test set.

The performance of the LM in ASR tasks is measured using the word error rate (WER). WER is calculated by comparing the reference (correct sentence) and the hypothesis (predicted sentence). Two sentences can be aligned with dynamic programming for string alignment to minimise the Levenshtein distance which is shown in Eq. 3.7.

$$\mathrm{WER} = \frac{S+D+I}{N} \times 100\%, \tag{3.7}$$

where S, D and I are the substitution, deletion and insertion error respectively. Specifically, 1-best hypotheses obtained using the LMs are scored against the reference transcription using the NIST Sclite scoring tool. In addition to the WER, sentence error rate (SER) (No. of incorrect hyp-ref pair divided by total number of hyp-ref pairs) is also provided by this tool.

3.5 Results and Discussions

3.5.1 LM Perplexity Comparison

To begin with, individual LM performances measured using PPL on PTB dataset for 4-gram LM, basic RNNLM, GRU and LSTM LM are compared in Table. 3.3.

LMs	4-gram LM	basic RNN	GRU	LSTM (1L)	LSTM (2L)
dev	156.4	142.9	126.3	103.8	**96.7**
eval	148.9	133.4	126.3	97.5	**91.2**

Table 3.3 PPL for different kinds of LMs on penn-treebank. All LMs are trained on in-domain data only. The vanilla RNN, GRU and LSTM all have 512D word embeddings and 512D hidden states. both single (1L) and double layer (2L) LSTMs are tested.

Spliced sentence bunches with 12 BPTT steps and 64 batch sizes are used to train these neural networks. Shown from the table above, basic RNNLM provides a relative 9% improvement over the 4-gram LM. GRU LM which has much slightly more parameters gives another 10% relative improvement over the basic RNNLM. Finally, LSTM LM gives the best performance over different types of language models which is 30% better in perplexity relative to GRU LM. To further corroborate that the improvements are consistent, these LMs are also tested on the AMI dataset shown in Table. 3.4.

LMs	Dev	Eval	Average Training Speed
4-gram LM	102.8	88.4	N/A
basic RNN	98.8	83.4	17ms/batch
GRU	81.9	69.1	20ms/batch
LSTM (1L)	71.4	63.3	21ms/batch
LSTM (1L)+4-gram ($\lambda = 0.8$)	**70.1**	**62.1**	N/A

Table 3.4 PPL for different kinds of LMs on AMI dataset. The vanilla RNN, GRU and LSTM all have 256D word embeddings and 256D hidden states. λ is the interpolation weight in Eq. 2.18. The training speeds are all recorded on air202 machine.

Consistent improvements are found by using LSTM LM, and the training speed of the LSTM model is reasonable, with only 18% relatively slower than the basic RNN model. However, as it uses much more space on the GPU than the RNN, and when the size of LSTM becomes larger, training speed decreases more significantly than the RNN model. Additionally, further improvements are found by interpolating the LSTM LM with the 4-gram LM. The interpolation weight λ is tuned manually and the best value is found at 0.8.

One of the important factor that influences the long-term dependency is the number of BPTT steps. In order to determine the best BPTT step for future expriments, the PPL against BPTT steps for LSTM LM is plotted in Fig. 3.4.

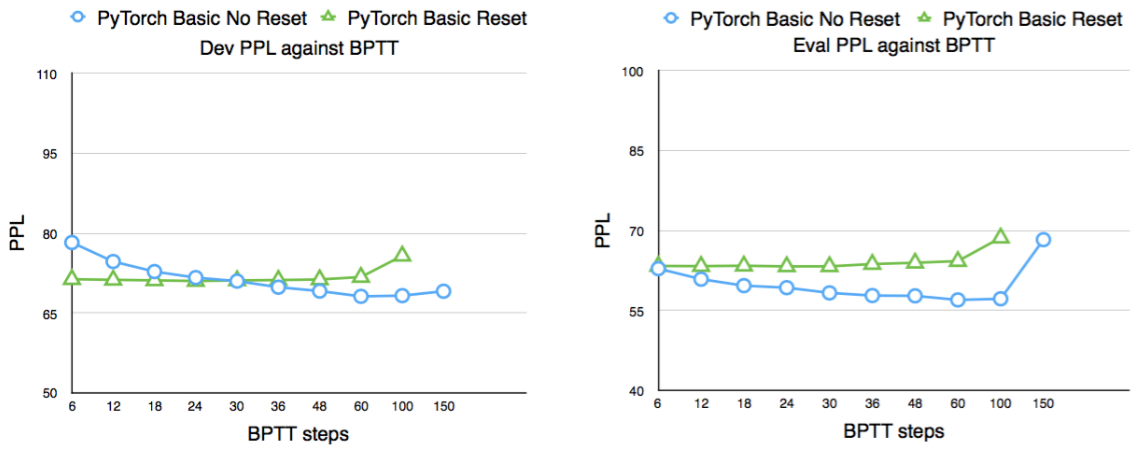


Fig. 3.4 PPL variation against BPTT steps on AMI. The LSTM LM uses 256D hidden states and 256D word embeddings. The green line represents the LSTM with sentence boundary resetting, and the blue line represents the one without resetting.

Under non-resetting scheme where the hidden state is carried on to the next utterance at each sentence boundary, a decrease in PPL is found until around 60 steps. This indicates that the gradient calculated through LSTM itself has the ability to influence around 60 words, and within 60, the longer the BPTT is, the better the long-term dependencies are. However, this reaches a plateau after 60 words and increasing the number of BPTT steps only marginally improve the PPL. Finally for very large BPTT step, PPL becomes worse as gradient estimation is inaccurate and the SGD algorithm is also hard to be effective on AMI corpus. On the other hand, with reset the plateau is reached at a much smaller BPTT step number as it is also affected by the average length of utterances. Therefore, for the resetting scheme, 12 BPTT step is used while for the non-resetting scheme. 64 BPTT step is used.

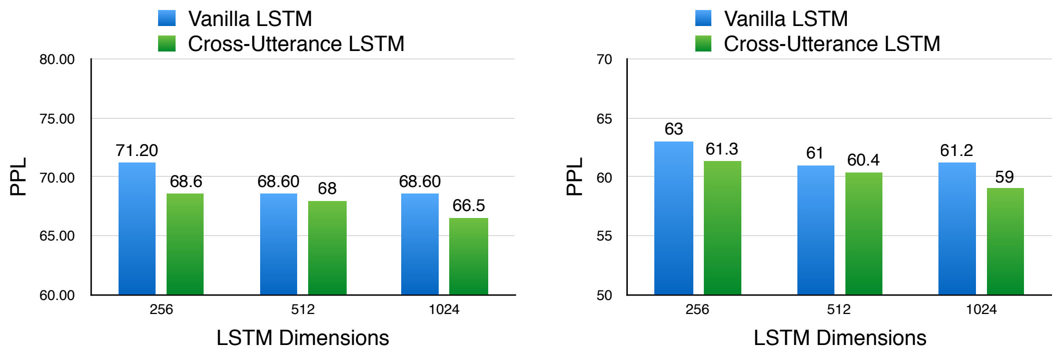


Fig. 3.5 PPL variation against LSTM size on AMI. Also comparison of the vanilla LSTM and the cross-utterance LM. Sizes on the x-coordinate refers to both the hidden states and the word embeddings. Context ranges from -3 to +3 utterances.

The first experiment with the cross-utterance LM uses the fully-connected layer as the context extractor. The context covers a range from previous 3 utterances to the future 3 utterances, and by varying the size of the LSTMs, PPL variations is shown in Fig. 3.5. The context vector is always kept the same size as the word embedding in this experiment. Similar to the effect of increasing the number of layers, by widening the layer of the LSTM, improvements are found in PPL. These improvements are consistent with different network sizes. For faster experiments and suitability of the size to the size of the training corpus, 256D LSTM will be used.

Next, different extractors are compared. As the fully-connected layer can not be too wide to cover many utterances, only previous and future 3 utterances are used to perform this experiment. The context coverage for the other two extractors ranges from previous 7 and future 7 utterances. The same first-level LM is used which is the 256D LSTM, and the final context vector is 128-dimensional for all three extractors. PPL results for cross-utterance LMs with these three extractors are shown in Table. 3.5

LMs	Dev	Eval	Average Training Speed
LSTM (1L)	71.4	63.3	21ms/batch
fully-connected	68.6	61.3	35ms/batch
Self-attention	68.3	60.9	51ms/batch
hierarchical LSTM	**68.0**	**60.6**	48ms/batch

Table 3.5 PPL for different kinds of context extractors on AMI. Average training speed is measured on air202 machine. Each mini-batch contains 64 data streams with 12 BPTT steps.

Note the training speed does not include the processing time for data arrangements which will take another 3 minutes to load the utterance embeddings at the start of training. Taking that into account, there are 1200 mini-batches in the AMI meeting corpus, hence the total time for training one epoch is around one minute for the cross-utterance LM.

On LM performance only, all cross-utterance LMs demonstrated improvements over the vanilla LSTM. The hierarchical LSTM extractor performs marginally better (rel. 5%) than the other two extractors (rel. 4% and 4.5% respectively), and the fully-connected layer extractor has the fastest training speed. It is also worthwhile to mention that, the fully-connected layer used for extraction, as well as to map the other extractor outputs to a lower dimensional space, use rectified linear unit (ReLU) activation function. Other activation functions have been tested such as Sigmoid and Tanh, but ReLU offers a much better result than the others.

Before moving on to the comparison of WER, the negative log-probability is plotted for several example utterances to demonstrate the discrepancy between different LM predictions in Fig. 3.6. The vanilla LSTM, vanilla LSTM with utterance-level shuffling before the start of each epoch, and the cross-utterance LSTM with fully-connected extractor are used. The vanilla LSTM gives similar performance in PPL with and without data shuffling.

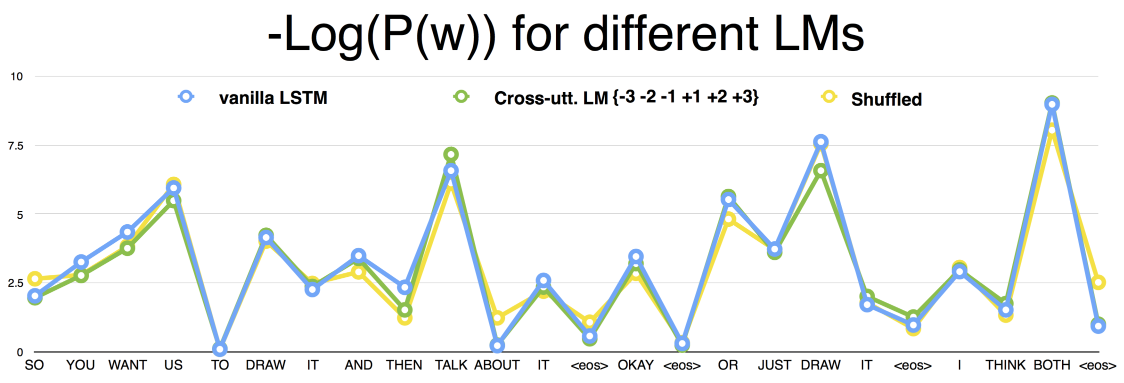


Fig. 3.6 Negative log-probability from different LM predictions for example utterances.

In general, higher probabilities (lower height in the figure shown) appears at more frequent words such as "TO" or "<eos>", and LM gives lower probabilities for less frequent words. Also for short utterances, vanilla LSTM and cross-utterance LM give very similar predictions, while for longer utterances, the cross-utterances gives higher probabilities.

3.5.2 WER with N-best Rescore

In order to see if the performance improvements in PPL can be transferred to the reduction in WER, a 50-best list for the dev set obtained from the first-pass decoding with a tri-gram is rescored with different LMs. The WERs using acoustic model trained with CE criterion are shown in Table. 3.6 and WERs using MPE criterion are shown in Table. 3.7.

RNNLMs	Substitution	Deletion	Insertion	Sentence Error	WER
tri-gram	14.9	7.0	5.0	61.0	26.9
vanilla LSTM	12.5	6.5	4.5	56.7	23.5
cross-utt. FC	12.4	6.6	4.4	56.6	23.4
cross-utt. FC (True)	12.4	6.4	4.5	**56.5**	**23.3**
oracle (HResults)	8.3	3.5	2.4	35.6	14.2

Table 3.6 WER break-down comparison between different RNNLMs on dev set. Acoustic model trained using CE-criterion. (True) means utterance embeddings are obtained using the true context. FC represents the fully-connected layer for context extractor.

RNNLMs	Substitution	Deletion	Insertion	Sentence Error	WER
tri-gram	13.5	8.0	4.1	59.2	25.7
vanilla LSTM	11.1	8.0	3.7	55.6	22.8
cross-utt. FC	11.0	8.0	3.6	55.5	**22.6**
cross-utt. FC (True)	11.0	7.9	3.6	55.5	**22.5**
CUED-RNNLM	11.6	8.0	3.8	56.9	23.4
Kaldi*	N/A	N/A	N/A	N/A	22.8

Table 3.7 WER break-down comparison between different RNNLMs on AMI dev set. Acoustic model trained using MPE-criterion. FC represents the fully-connected layer for extractor.

Note in Table 3.6, the oracle value is obtained using HResult instead of Sclite which refers the best result that can be achieved with this 50-best list. Also in Table 3.7, CUED-RNNLM uses the same size as the vanilla LSTM, and rescores on the same 50-best list. Kaldi numbers are taken from [23] which is rescored based on a different acoustic model. The language model scale factor μ is tuned and fixed to be 16 which is proved to be the most suitable value for all different RNNLMs in PyTorch implementation. Comparing the numbers across the two tables, using sequence discriminative training criterion reduces the error rate by 0.6 to 0.7 in absolute value. Consistent improvements have been found in WER for cross-utterances LMs with fully-connected layer as the context extractor.

In Table. 3.6, using cross-utterance LM with hypothetical utterances as context reduces the WER by 0.1 in absolute value, and using the true context gives a further 0.1 absolute reduction. The true context slightly affects the WER, but as the difference is not very large, the utterance embeddings are relatively robust to the acoustic errors if the utterance embeddings are not jointly trained with the second-level LM. In Table. 3.7, similar improvements are also found, and the PyTorch implementation of LSTM LM outperforms the CUED-RNNLM results, and achieves a similar WER as that achieved by Kaldi.

RNNLMs	Substitution	Deletion	Insertion	Sentence Error	WER
vanilla LSTM	11.1	8.0	3.7	55.6	22.8
cross-utt. FC	11.0	8.0	3.6	55.5	**22.6**
cross-utt. atten.	11.2	7.8	3.7	55.4	22.7
cross-utt. hier.	11.3	8.0	3.6	55.7	22.9

Table 3.8 WER break-down comparison between different extractors on dev set. Acoustic model trained using MPE-criterion. FC represents the fully-connected layer for extractor. Atten. refers to self-attentive strucutre and Hier. refers to the hierachical LSTM structure.

Furthermore, experiments are performed on different extraction networks as well which is shown in Table. 3.8. Inferred from the results, the best WER is achieved by the fully-connected layer even though the hierarchical LSTM has the best PPL. The hierarchical LSTM has the highest WER which does not match its low perplexity. This is probably because it over-fits on the true context, and hence experiments should be done on rescoring with the true context, which will be left for future investigations. However, the sentence error rate is higher for the fully-connected extractor than that for the self-attentive extractor. Results on the eval set are shown in Table A.1 in Appendix A.

The final experiment with fixed utterance embeddings is to investigate the influence of context coverage on PPL and WER, and hence to choose a suitable range of context. As the WER only changes at 0.1 level, the PPL is plotted against different context coverage. The self-attentive extractor and the fully-connected extractor are plotted as shown in Fig. 3.7.

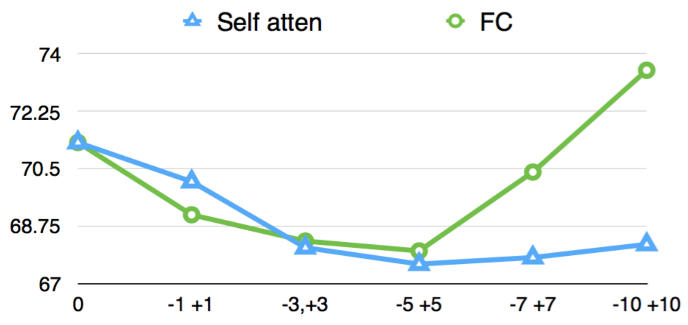


Fig. 3.7 PPL variation against context coverage on AMI. 0 represents no context which is the baseline vanilla LSTM LM. -3 +3 represents previous 3 and future 3 utterances are used.

Inferred from the plot below, the best performance is achieved when 5 utterances from the past and the future are used for both extractors. The fully-connected layer is limited by the width of that layer when the context coverage becomes large. It can mitigated using TDNN structure [38] which uses fully-connected layers in a dilated way.

3.5.3 Summary

The main findings from the experiments can be summarised as follows.

- Consistent improvements in PPL are found by using the basic RNNLM instead of the n-gram model. Further improvements are obtained using LSTM structure.
- Interpolation of the LSTM LM with a 4-gram LM produces better PPL than using LSTM alone. The interpolation weight can be manually adjusted.
- The cross-utterance LMs with different context vector extraction networks outperforms the vanilla LSTM model in PPL.

- The hierarchical LSTM structure achieves the lowest PPL.
- Improvements are also achieved in WER when RNNLM is introduced to replace the n-gram LM. WER is also improved by using the context vector extracted from either a fully-connected layer, a self-attentive layer and a hierarchical LSTM structure.
- The hierarchical LSTM with the best PPL obtains the worst WER among different extractors. Fully-connected layer performs the best in WER.
- Suitable context coverage is obtained for fully-connected and self-attentive extractor.

Chapter 4

Joint-training of Cross-Utterance RNNLMs

Joint-training refers to the process of optimising different components in a deep neural network system under the same criterion. It requires all parameters in different components to be updated during training instead of the two-stage implementation in the previous chapter. Joint-training is usually used to mitigate the criteria mismatch when a task is split into many stages. In this case, the training of the first-level LM is to predict the next word rather than extracting a better utterance representation. However it is the utterance representation that is used in the second-level LM. While the second-level LM is trying to learn how to extract useful information from the context, the first-level LM is also desired to learn how to provide such information. Therefore, joint-training algorithm design is necessary to the two-level cross-utterance LM in order to get further improvements.

The joint-training algorithm design is detailed in this chapter. To begin with, the joint-training algorithm is described with particular attention to implementation details and training tricks. Then, under the joint-training framework, 3 different types of context arrangements are used to extract context information with different context extraction networks. Experiments are performed on the AMI meeting dataset.

4.1 Joint Training Algorithm and Implementation Details

The training data arrangement for the second level still follows the spliced sentence bunch arrangement. As before, for each chunk of data, a copy that contains the utterance indices associated with each word in the chunk is produced. During the data preparation stage, an utterance hash table is constructed for each data file which can be indexed very fast. Then, when processing a specific chunk, all utterances that will be used for context vector will be collected from the utterance hash table, and will be forwarded using the first-level LM to get the utterance embeddings. These utterance embeddings are used for the second-level LM as before. Because gradients from the second-level LM will accumulate for each utterance embedding every time this embedding is used, gradients can be back-propagated to the first-level LM via the gradient tensor of each utterance embedding.

Algorithm 1 Joint Training Algorithm

```
Load pre-trained first-level LM and randomly initialise second-level LM
Initialise learning rate, gradient clip, dropout rate for both LMs separately
for file in training data files do
    Read file in and convert to a word index tensor: training data
    Fill in the hash table of utterances: utterances
    Assign utterance index to each word
    Expand utterance index to get context set for each word
    Split the training data into chunks (mini-batches)
    for chunk in training data do
        Collect indices of utterance embeddings needed in chunk
        Clear the first level LM parameter gradients
        Initialise the hash table of embeddings: embeddings
        for index in indices do
            Initialise the first-level LM hidden states
            Forward the first-level LM with utterances[index]
            Store final hidden state into embeddings[index]
        end
        Initialise auxiliary_input tensor
        for word in chunk do
            Get utterance indices associated to word
            Fill the auxiliary_input[word] with embeddings[indices]
        end
        Clear the second level-LM parameter gradients
        Forward the sencond-level LM with chunk and auxiliary_input
        Calculate the cross-entropy loss
        Calculate gradients for both LMs
        Update the second level LM parameters
        Update the first level LM parameters
    end
end
```

As it is not an off-the-shelf algorithm, a lot of adjustments and optimisation tricks have been tried and incorporated into this training algorithm. There are alternatives in three parts in the above algorithm that will be discussed in this section: the choice of utterance embeddings, the updating scheme and the pre-training for the first-level LM.

There are two choices of utterance embeddings in general if the first-level LM uses the LSTM structure. One is the memory cell which acts like the memory in a computer, and another is the output hidden vector which decide what to fetch from the memory and what to write to the memory cell to make the prediction. In fixed embedding training algorithm, the memory cell is used as the utterance embedding since it captures the most history information. The extractor takes over the responsibility to learn how to extract useful information. However, if the two LMs are jointly trained and hence the output hidden vector can be jointly optimised, it is better to leverage the design of the LSTM structure and to train the output hidden vector to get utterance representations from the memory. Then, if the concatenation of all the output hidden vectors is too large, a simple linear transformation could be used to map them down to a compact representation which is used as the context vector. To further avoid the bias of these output hidden vectors towards nearer words, not only the final but also the middle output hidden vector in an utterance could be used.

This system contains two parts that are used for different tasks, and the first-level LM has much more samples per update than the second-level LM. Therefore, in order to better optimise both networks, two networks use different set of hyper-parameters such as learning rate, dropout and weight decay. Furthermore, the number of samples seen by the first-level LM might be different from batch to batch as the number of utterances and utterance lengths may be different. The gradient is thus accumulated for several epochs and then update so that the number of samples per update is roughly constant. Last but not least, interleaved training which has been used for multi-task learning is also adopted in this task for the first several epochs. This means one LM parameters are fixed while the other one is being updated. This is necessary when pre-training of the first-level LM is applied.

Pre-training of neural network is widely used for better model initialisation, and the pre-training followed by fine-tuning pipeline has been gradually popular in LM related researches such as the word embeddings in BERT [24]. It refers to the process of training the model on some tasks, and then take the parameters from the trained model as the initialisation for another task. In this case, the first-level LM can be trained for 1 or 2 epochs on language modeling task, and take its parameters to initialise the cross-utterance LM. When joint-training the two-level LM, the pre-trained parameters are firstly fixed for one or two epoch for the second-level LM to be reasonably good to use the information provided by the first-level. This interleaved training will last for 3 to 4 epochs until both networks are reasonably good, and then they will be optimised (fine-tuned) simultaneously.

4.2 Context Arrangements

The algorithm described in the previous section is implemented with the sentence structure retained, but it can be generalised to many context arrangements. This section will discuss 3 different context arrangement methods that this training algorithm with hash tables can adopt. The first one retains the sentence structure, and the other two ignores the sentence structure but looking at a number of words surrounding the current utterance.

4.2.1 LSTM-based Utterance Embeddings

This data arrangement, referred to as the utterance embeddings, and its training strategies have been discussed in previous section. The illustration of generating the context vector using these utterance embeddings is shown in Fig. 4.1.

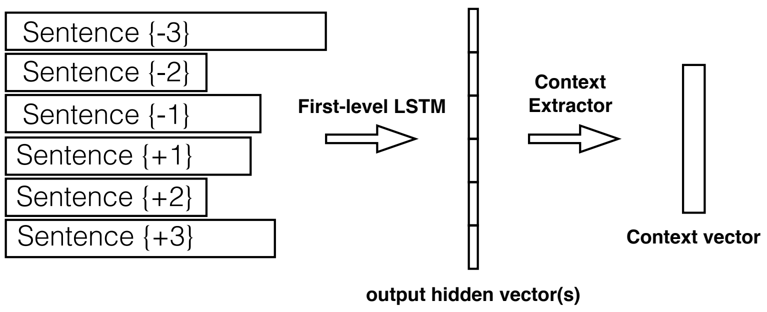


Fig. 4.1 Context representation using the utterance embeddings where the utterance structure is retained. Here previous 3 utternaces and future 3 utterances are used.

The context extractor could be a simple linear layer or a self-attentive layer. The most significant advantage of this setting is that it retains the sentence structure. Each embedding contains the information of one single complete utterance without mixed meanings from other utterances such as the one introduced next. On the other hand, the biggest problem with this training scheme is the data processing time and training speed. The context used for each chunk could be temporarily stored during the first epoch, and then used for the rest of the epochs. This speed up the processing time by 50% for the later epochs. However, because of the variable length of different utterances, it is hard to exploit the GPU parallel processing. It is possible to used `pad_packed_sequence` and `pack_padded_sequence` in PyTorch which helps to exploit better usage of GPU, but these methods require the utterances to be sorted, which adds onerous processes to the data preparation stage. Another drawback for this arrangement is that the number of samples in each mini-batch for the first-level LM may vary significantly. Accumulation of gradients is used to mitigate this problem.

4.2.2 LSTM-based Segment Embeddings

In order to exploit the GPU parallelisation and to further address the variable sample size problem, the context vector could also generate from segment embeddings extracted for a fixed number of words surrounding the current utterance, as illustrated in Fig. 4.2.

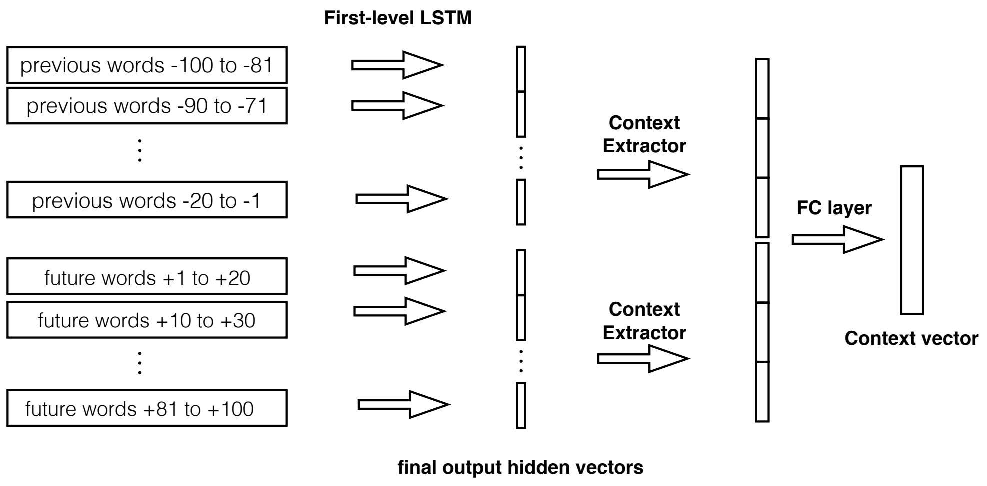


Fig. 4.2 Context representation using the segment embeddings where the sentence structure is ignored. Context coverage ranges from the previous 100 words to future 100 words, with each segment of 20 words and 10 words overlap.

This arrangement takes the advantage of the fact that useful information in our task is often some key words or phrases in the surrounding utterances. These could be extracted via the output hidden vector from the memory cell if jointly trained as discussed before. Therefore, the boundaries are treated as normal words, or can even be ignored. Hence, instead of getting a number of surrounding utterances, this modified version tries to get a fixed number of surrounding words for the current utterance, split them into segments and feed them into the first-level LSTM LM. The parameters are shared among all first-level LSTMs for all segments, and the hidden states are reset to zero at the start of each segment. Furthermore, there can be a fixed number of overlapping words between two adjacent segments.

This arrangement is able to be processed in parallel easily since all segments have the same length, and the number of samples processed by the first-level LM is also the same for each mini-batch. However, it does not retain the sentence structure, and it only relies on the output hidden vector to find useful information from the memory rather than explicitly express the relative importance of different words. The embeddings might contain mixed information and have a bias towards the words closer to the current utterance. Increasing the number of overlapping words might mitigate this problem, but this requires more segments to cover the same range of context, hence makes the extractor computationally more expensive.

4.2.3 Self-attentive Segment Embeddings

In order to better exploits the segment structure and to explicitly express the importance of important words and phrases in each segments, a multi-head self-attentive layer taking the same form as described in Sec. 3.2 is used. The arrangement is shown in Fig. 4.3.

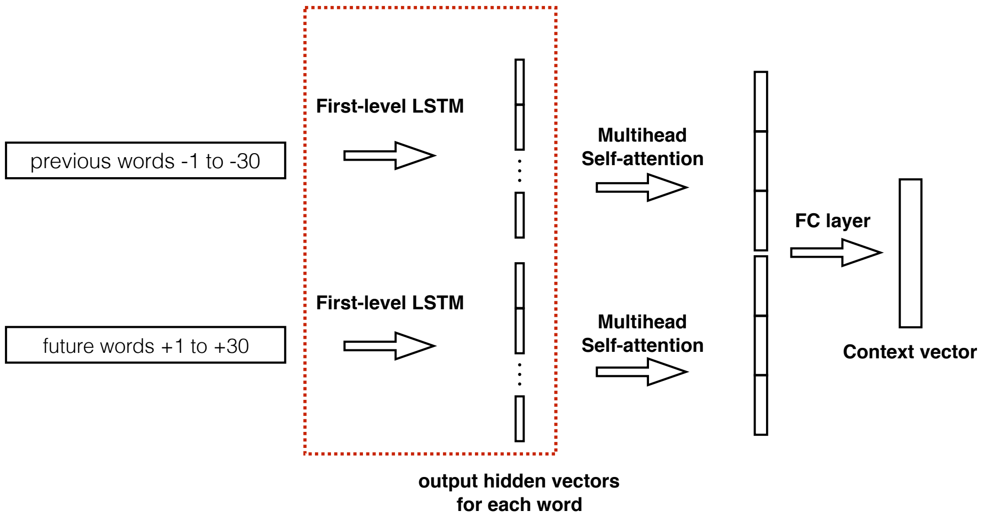


Fig. 4.3 Context representation using the self-attentive segment embeddings where the sentence structure is ignored. Context coverage ranges from the previous 30 words to future 30 words, with each segment of 30 words and no overlap.

The segment is passed through an LSTM and instead of using only the final output hidden vector as before, output hidden vectors are collected and fed into the self-attentive layer. The multi-head output of the attention layer are then concatenated and sent to a fully-connected layer to get a compact representation. The first-level LSTM shown in the dashed box is to encapsulate the positional information among the words, but some recent researches established methods for positional encoding [25]. Through the multi-head self-attentive layer structure, the relative importance of different words in each utterance can be directly reflected by the values in each annotation vector. By forcing the annotation vectors to be dissimilar using the penalty term, diverse information for each segment could be extracted, and hence it handles longer segments.

4.3 Experiments

Experiments are performed based on the fully-connected layer or self-attentive layer context extraction. Hierarchical LSTM becomes extremely deep if jointly trained with the first-level LM being another LSTM, hence the appropriate and efficient joint-training algorithms for hierarchical LSTM structures will be left for future investigation. It mainly compares the improvements obtained by jointly train cross-utterance LMs. The presentation of the result will be divided into three parts according to the data arrangements in the previous section, and with a summary of the best results obtained from these arrangements.

4.3.1 Utterance Embeddings

This section first shows the effect of different training techniques and hyper-parameters on PPL, which describes how the gain in PPL is obtained. The techniques are tabulated below.

①	Utterance embedding: Use final memory cell
②	Utterance embedding: Use final output hidden vector
③	Utterance embedding: Use final and middle output hidden vectors
④	Updating Scheme: Simultaneously update both LMs
⑤	Updating Scheme: Interleaved training
⑥	Updating Scheme: Accumulate 10 mini-batches then update
⑦	Hyper-parameters: Same learning rate for both levels
⑧	Hyper-parameters: Smaller learning rate for the first-level LM

Table 4.1 Different training techniques and facts applied to the joint-training algorithm.

The vanilla LSTM and the cross-utterance LM with fixed embeddings will be used as baseline systems. Previous 3 utterances are incorporated for quick experiment. Because the training speed for the utterance embedding joint-training algorithm takes much longer to train, it is not feasible to try every combination of the techniques shown above. However, the results shown in Table. 4.2 are still enough to reflect how the system should be jointly trained. Note that if the accumulation across mini-batches is used, the learning rate needs to be further adjusted. Therefore, ⑥ implies ⑧ is also used.

LMs	Dev	Eval	Average Training Speed
LSTM (1L)	71.4	63.3	21ms/batch
fully-connected cross-utterance LM	68.6	61.3	35ms/batch
① + ④ + ⑦	69.7	62.2	680 ms/batch
① + ⑤ + ⑦	68.6	61.5	-
① + ⑤ + ⑥	67.9	61.0	-
② + ⑤ + ⑥	68.8	62.3	-
③ + ⑤ + ⑥	**67.3**	**60.6**	-

Table 4.2 PPL for joint-training with different techniques on AMI. Average training speed is measured on air202 machine. Hyphen represents the same value as above. Each mini-batch contains 64 data streams with 12 BPTT steps.

The size of the network used here is the same as before with single layer LSTM which has 256D word embeddings, 256D hidden states and 128D context vector. Each utterance embedding is of 256D as well. The average processing time for each mini-batch is 680 ms, hence for 1200 mini-batches, it requires 13.6 minutes to process one epoch. Next, using final and middle output cell, interleaved training and gradient accumulation over mini-batches, wider context coverage is explored and better results have been obtained shown in Table. 4.3.

LMs	Dev	Eval	Average Training Speed
vanilla LSTM	71.4	63.3	21ms/batch
fully-connected cross-utterance LM	68.6	61.3	35ms/batch
Joint-trained cross-utterance LM	**65.3**	**58.3**	1500 ms/batch

Table 4.3 PPL for joint-training with previous and future 5 utterances on AMI. Speed measured on air202. Each mini-batch contains 64 data streams with 12 BPTT steps.

Though 9% relative improvement and 6% relative improvement in PPL compared to the vanilla model and the fixed embedding model have been achieved respectively, this training speed is not suitable for large scale data set because the processing time for one epoch is 30 minutes. For medium scale data sets such as Switchboard and Fisher which contains 27 million words, it requires a whole day to train this rather small model.

Finally, the joint-trained LM is used for 50-best rescoring with acoustic model trained using CE criterion. The result is shown in Table. 4.4 where another 0.1 absolute reduction in WER and 0.2 absolute reduction in sentence error have been found.

RNNLMs	Sent. Error (dev)	WER (dev)
vanilla LSTM	56.7	23.5
cross-utt. FC	56.6	23.4
joint-trained cross-utt.	**56.4**	**23.2**

Table 4.4 WER comparison between different LMs on AMI dev set. Acoustic model trained using CE-criterion. Joint training uses the best techniques.

4.3.2 LSTM-based Segment Embeddings

Experiments on segment embeddings focus on the training speed and the effect of context coverage. PPLs for different context coverage and overlappings are shown in Table 4.7.

Context	Segments	Overlap	Dev	Eval	Average Training Speed
$\pm$ 36 words	2	0	64.3	57.3	215 ms/batch
$\pm$ 36 words	4	0	65.5	59.1	297 ms/batch
$\pm$ 72 words	4	0	64.8	58.0	263 ms/batch
$\pm$ 72 words	8	18	**63.6**	**57.5**	300 ms/batch
$\pm$ 108 words	12	18	64.5	58.6	326 ms/batch
vanilla LSTM			71.4	63.3	21ms/batch
FC. cross-utt. LM			68.6	61.3	35ms/batch

Table 4.5 PPL for joint-training with segments on AMI. Same mini-batch size and same machine as before. segment length = context/segments + overlap. e.g. $\pm$ 72 words with 8 segments and 18 overlap has length $144/8+18=36$ words.

Note that because the number of samples processed by the first-level LM for each mini-batch is now fixed and is more than the second-level LM, a small learning rate for the first-level LM and the gradient accumulation is not necessary anymore. The same network size is used for these experiments. A lot more experiments on different context coverage have been conducted but only the ones showing valuable results are presented here. Under the best context coverage and overlap settings, the LSTM-based segment embeddings outperformed the vanilla LSTM by a relative 11% reduction in PPL. The total 1200 mini-batches requires 6 minutes to process one epoch, which is much faster than the utterance embedding arrangement since all segments are processed in parallel by the GPU. Then the best setting is used to rescore 50-best lists on AMI dev set as before, and the WER results for CE training and MPE training are shown in Table 4.6. Best performances are achieved by the joint-trained cross-utterance LM with segment embedding for both CE (0.2 improvement) and MPE (0.1 improvement) acoustic models.

RNNLMs	Sent. Error (dev)	WER (dev)	Sent. Error (eval)	WER (eval)
vanilla LSTM CE	56.7	23.5	55.6	24.2
fixed cross-utt. CE	56.6	23.4	55.5	24.1
seg. embeeding CE	**56.4**	**23.2**	**55.3**	**24.0**
vanilla LSTM MPE	55.6	22.8	54.8	23.5
fixed cross-utt. MPE	55.5	22.6	54.6	23.4
seg. embedding MPE	**55.3**	**22.5**	**54.3**	**23.3**

Table 4.6 WER comparison between different LMs on AMI.

4.3.3 Self-attentive Segment Embeddings

Finally, experimental results on the self-attentive segment embeddings are presented in this section. There are always two attention layers applied to the past and future context separately, and hence there are always two segments to perform attention on. PPL for context coverage with ± 50 and ± 100 words are shown in Table, where experiments have also been performed on the number of heads (i.e. the number of annotation vectors in Eq. 3.2).

Context	No. of heads	Dev	Eval	Average Training Speed
± 50	1	64.0	57.8	312 ms/batch
± 50	3	63.8	57.9	334 ms/batch
± 100	1	63.0	57.3	356 ms/batch
± 100	3	**62.6**	**56.8**	362 ms/batch
± 100	5	62.7	57.0	370 ms/batch
vanilla LSTM		71.4	63.3	21ms/batch
FC. cross-utt. LM		68.6	61.3	35ms/batch

Table 4.7 PPL for joint-training with self-attentive segment embeddings on AMI. Same mini-batch size and same machine as before. Penalty term scaling $\mu = 0.0001$.

Again small network size with 256D word embeddings and 256D hidden states are used. 12% relative improvements have been found by using the self-attentive layer compared to the baseline vanilla LSTM LM. Not much improvements have been found by including more heads, hence it is also interesting to plot some annotation vectors against the words it covers to see what information the model focuses on. The current text for these annotation vector is: `SO THAT MEANS BASICALLY NEXT TUESDAY`.

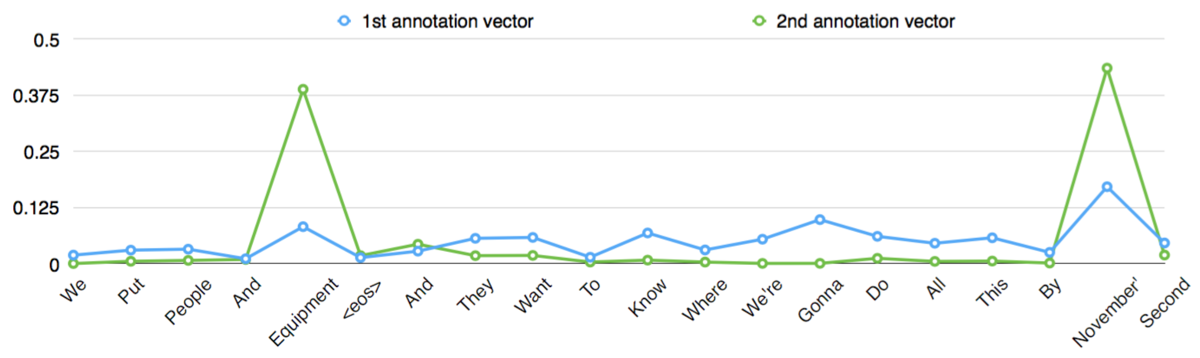


Fig. 4.4 Two annotation vectors covering the previous 20 words

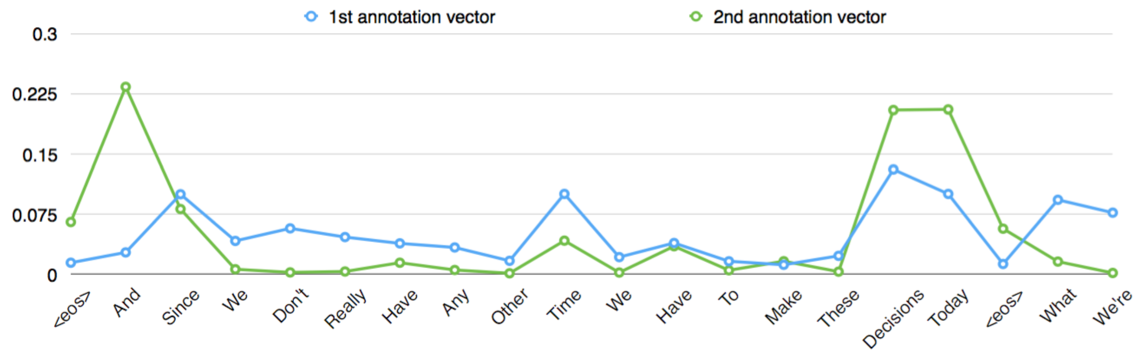


Fig. 4.5 Two annotation vectors covering the future 20 words

As this utterance basically talks about a time to do something, the annotation vector pays attention to words like "today", or "November". It also gives attention to connecting words such as "and" and "since". However, the two annotation vector do not differ much in their major focuses. The minor difference is that the second vector tends to get other words incorporated as well. Therefore, the scale for the penalty term is increased, but the perplexity becomes even worse as the main loss function influence is relatively attenuated. Therefore 3-head self-attentive layer is used to extract the context information from the segments. This cross-utterance LM is also used to rescore the 50-best list, and results are shown in Table 4.8.

RNNLMs	Sent. Error (dev)	WER (dev)	Sent. Error (eval)	WER (eval)
vanilla LSTM CE	56.7	23.5	55.6	24.2
fixed cross-utt. CE	56.6	23.4	55.5	24.1
self-atten. CE	**56.5**	**23.3**	**55.4**	24.1
vanilla LSTM MPE	55.6	22.8	54.8	23.5
fixed cross-utt. MPE	55.5	22.6	54.6	23.4
self-atten. MPE	**55.5**	**22.5**	**54.5**	**23.3**

Table 4.8 WER comparison between different LMs on AMI set.

4.3.4 Summary

All results obtained from different data arrangements are summarised in this section for AMI dev set. In addition to the small LMs (256D), PPL and WER are also tested for a larger network (512D). Furthermore, interpolation with the tri-gram LM which was used for the first-pass decoding is also implemented for the second-pass decoding. The PPL results for the systems used for rescoring are shown in Table 4.9. WER results for CE acoustic models are shown in Table 4.11, while that for MPR acoustic models are shown in Table 4.10.

System	Network Size (word emb:hidden:context)	Context Coverage	PPL
vanilla LSTM small	256:256:N/A	N/A	70.5
FC cross-utt. small	256:256:128	±3 utterances	68.5
utterance emb. small	256:256:128	±5 utterances	65.2
segment emb. small	256:256:128	±72 words	64.3
attention-based small	256:256:128	±100 words	**62.6**
vanilla LSTM large	256:512:N/A	N/A	69.5
utterance emb. large	256:512:256	±5 utterances	64.1
segment emb. large	256:512:256	±72 words	63.6
attention-based large	256:512:256	±100 words	**61.8**

Table 4.9 PPL for different LMs on AMI dev set. Small refers to 256D LMs and large refers to 512D LMs. Context coverage uses the best in the previous experiments.

System	Sentence Error	WER
vanilla LSTM small	55.6	22.8
FC cross-utt. small	55.5	22.6
LSTM-based segment emb. small	**55.3**	**22.5**
vanilla LSTM large	55.6	22.6
utterance emb. large	55.1	**22.3**
LSTM-based segment emb. large	**54.9**	**22.3**
attention-based large	55.3	22.4
vanilla LSTM + trigram	55.0	22.3
LSTM-based segment emb. large + trigram	**54.5**	**22.0**

Table 4.10 WER for different LMs. Acoustic model trained with MPE criterion. +trigram means interpolating with the trigram used in the first-pass decoding.

System	Sentence Error	WER
vanilla LSTM small	56.7	23.5
FC cross-utt. small	56.6	23.3
utterance emb. small	56.4	**23.2**
LSTM-based segment emb. small	**56.2**	**23.2**
attention-based emb. small	56.3	23.3

Table 4.11 WER for different LMs. Acoustic model trained with CE criterion.

To summarise, consistent improvements have been found using different sizes of LSTM and using interpolation. However, self-attentive layer segment embedding extraction which gives the lowest PPL does not give the lowest WER. One possible explanation is that the prediction using the self-attentive segment embeddings are too sharp on certain words, hence when acoustic errors are added to not only the context but also the current utterance. One possible way to mitigate this is to use acoustic error sampling.

Chapter 5

Acoustic Error Sampling

Error sampling is a broadly used technique in language modelling which substitutes the true word with other words according to a certain probability distribution. Acoustic error sampling here refers to the sampling technique that samples from some possible acoustic error distributions in order to mitigate the mismatch between training and application of the LM. In this chapter, the motivation for the error sampling algorithm will be discussed where recent researches in error sampling techniques are briefly accounted, followed by the implementation. Due to the limited amount of time, only some preliminary experiments will be described in this chapter.

5.1 Motivations for Error Sampling

As the language modelling task is performed on text only, the true history or context is always available. However in the downstream tasks such as ASR, neither true history within the utterance nor the true context in the auxiliary input is guaranteed. Therefore, in order to mitigate this discrepancy, an acoustic error sampling method is proposed which incorporates the possible acoustic errors that might occur in the first-pass decoding hypotheses.

There are mainly two advantages to perform the error sampling: First, the error sampling prevents the language model from being over fit to the true context, and hence the language model generalises better on the rescoring task. Second, it augments the data as each utterance now yields multiple versions. Hence more epochs can be run and larger models can be used.

Many forms of error sampling has been proposed by recent researches in language modelling. One of the most successful methods is the scheduled sampling [28] which, during training, samples the Softmax output layer to substitute the true label. This error sampling technique provides the model-generated token for the next step prediction instead of the true one. It also controls whether the model uses the true one or the generated one hence controls the proportion of the erroneous labels. The distribution itself is conditioned on the previous observations, and this technique is useful for text generation tasks.

However, the distribution of the Softmax layer covers the whole vocabulary, while acoustic errors have a constrained shortlist of words which have either close pronunciations or meanings from the first-pass decoding. Therefore, [29] uses some distance measurements of pronunciation similarity to decide the distribution to be sampled from. However, this method is still based on theoretical values instead of the real acoustic errors.

As the aim of the language modelling is to give accurate prediction for the true utterance even if the context contains acoustic errors, sampling of acoustic errors will be performed on the context before each epoch. A method that uses the unigram model of the real acoustic errors obtained from first-pass decoding on the training set is proposed in the following section, where the training with sampling pipeline is described.

5.2 Training with Acoustic Error Sampling

After being scored with the Sclite toolkit, an error summary file will be given which has the count of each type of substitution, insertion and deletion. The preliminary implementation of the error sampling algorithm focuses on the substitution error. By parsing the error summary file, for each word a set of confusion pairs could be obtained. The error distribution is calculated by dividing each pair with the total number of occurrences of that word. For example for word "THEY" which occurs 1000 times in the training set, two confusion pairs ("THEY", "HE") and ("THEY", "SHE") are provided by the summary with counts 30 and 50 respectively, the probability of getting such replacements will be 0.03 and 0.05 respectively.

As the substitution error is already very rare for frequent words, the probability distribution of errors will be biased towards generating more errors by discounting on the frequency of the correct word. Then after one or two training epochs on the original data as before, the training set is first randomly sampled using the biased error distribution. Then the model is trained on the sampled train set. Besides, after each epoch the dev set is sampled and PPLs with and without error sampling will be calculated for the dev set.

It is also worthwhile to mention that with this sampling technique, it also allows the training without resetting at the sentence boundaries for cross-utterance LMs. As before, training the cross-utterance LM without resetting will bring the future utterance information to the model and hence will be memorised. However with the error sampling, the future utterance does not contain exactly the same words at all, but the meaning may not change much. Not resetting the hidden states accelerates the training speed and fixes the number of BPTT steps. Non-resetting rescoring can also be implemented as another kind of cross-utterance LM such as the one used in Microsoft transcription system [39], but it is out of the scope of this thesis.

The first insufficiency of this method is that it only uses uni-gram (confusion pair) models to obtain the error distribution, but a better distribution should also condition on the context. Consequently it is hard to incorporate insertions or deletions if the distribution is not conditioned on the context. This is the second drawback which may result in the self-attentive layer memorising the position of the words that are never changed, and giving high weights.

5.3 Experiments

The experiments are only performed on the jointly trained cross-utterance LM with segment embedding. The comparison in WER is shown in Table 5.1 where another 0.1 reduction is found compared to the jointly-trained one without error sampling.

RNNLMs	Sent. Error (dev)	WER (dev)
vanilla LSTM	55.6	22.6
cross-utt. FC	55.3	22.5
joint-trained cross-utt.	55.1	22.3
joint-trained + error sampling	**54.9**	**22.2**

Table 5.1 WER for LMs with or without error sampling on AMI dev set.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Recurrent neural network language models have been widely used for language understanding, translation, dialogue and speech recognition tasks. It provides a continuous-space representation for the history in a sequence that is used to predict the next word. Therefore RNNLMs achieve more accurate word prediction and outperform traditional n-gram LMs. However, the context the model can memorise and make use of is still limited because of the fixed dimensionality and error back-propagation through time algorithm. In particular, LM is combined with the acoustic model in ASR tasks usually at single utterance level.

In order to improve the long-term dependency of LMs and to make use of cross-utterance information in speech recognition tasks, a two-level cross-utterance LM is proposed. Different data arrangements, context extractor networks as well as joint-training algorithms are proposed. By comparing different types of LMs for their individual performances and performances in ASR systems, the conclusions of this work are drawn.

- Basic RNNLM outperforms the n-gram LM by a large margin in PPL as it releases the Markov assumption to get long-term dependency. It is also better in that it mitigates the data sparsity issue which occurs in the counting-based n-gram model.
- GRU and LSTM structures as two variations of the basic RNNLM are also experimented where the LSTM has the most number of parameters with the lowest PPL. LSTM structure is also shown to outperform the KN-smoothed 4-gram LM in WER.
- Cross-utterance LMs using fixed utterance embeddings with a fully-connected layer, a self-attentive layer and a hierarchical LSTM layer all achieves better PPL than the vanilla LSTM, with the hierarchical LSTM having the lowest PPL.

- Cross-utterance LMs with fully-connected layer and self-attentive layer for context vector extraction outperformed the vanilla LSTM in WER, while the hierarchical LSTM gives worse result.
- Joint-training of the two-level cross-utterance LM further improves the results in PPL and WER because both networks are being optimised under the same criterion.
- The best PPL is achieved by using the self-attentive segment embeddings for context representation, while the best WER is achieved by the LSTM-based segment embeddings. The fastest training speed is achieved by the LSTM-based segment embeddings which is 4 times slower tha the vanilla LSTM.
- Acoustic error sampling which acts as not only a regularisation method but also a data augmentation method can be applied before the start of each epoch, which releases the dependency of the cross-utterance LMs on the true context.

6.2 Future Work

- Investigations into the hierarchical LSTM and other extraction structures need to be explored, and the constraints on RNN-type LM can be released to include transformer architectures [25]. Besides, it is also an interesting research to implement the lattice rescoring for more complicated LMs such as this cross-utterance LM and the transformer LM. Furthermore, it is also desired to look into how the reduction in PPL can be transferred to the reduction in WER. One solution might be to incorporate the LM at an earlier stage to narrow the gap between the target which is the WER and the training criterion for LM.
- Another direction is to incorporate more efficient error sampling techniques especially for insertion and deletion so that the relative position between words can also be changed. This should be done using sequential sampling algorithms rather than unigram models. One possible way is to have another neural network to learn the error patterns in an recurrent fashion, and sample from that network. With error sampling, the cross-utterance LM can be trained without sentence boundary resetting to achieve a even lower PPL and WER.

References

[1] T. Mikolov, M. Karafiat, L. Burget, J.H. Cernocky, & S. Khudanpur, “Recurrent neural network based language model", *Proc. Interspeech*, Makuhari, 2010.

[2] A. Jaech, & M. Ostendorf, “Low-rank RNN adaptation for context-aware language modeling", *Proc. ACL*, Melbourne, 2018.

[3] I. Oparin, M. Sundermeyer, H. Ney, & J. Gauvain, “Performance analysis of neural networks in combination with n-gram language models", *Proc. ICASSP*, Kyoto, 2012.

[4] M. Sundermeyer, R. Schluter, & H. Ney, “LSTM neural networks for language modeling", *Proc. Interspeech*, Portland, 2012.

[5] X. Chen, Y. Wang, X. Liu, M.J.F. Gales & P.C. Woodland “Efficient GPU-based training of recurrent neural network language models using spliced sentence bunch", *Proc. Interspeech*, Singapore, 2014.

[6] X. Chen, T. Tan, X. Liu, P. Lanchantin, M. Wan, M.J.F. Gales, & P.C. Woodland, “Recurrent neural network language model adaptation for multi-genre broadcast speech recognition", *Proc. Interspeech*, Dresden, 2015.

[7] K. Li, H. Xu, Y. Wang, D. Povey, & S. Khudanpur, “Recurrent neural network language model adaptation for conversational speech recognition", *Proc. Interspeech*, Hyderabad, 2018.

[8] X. Chen, X. Liu, Y. Qian, M.J.F. Gales, & P.C. Woodland, “Cued-rnnlman open-source toolkit for efficient training and evaluation of recurrent neural network language models", *Proc. ICASSP*, Shanghai, 2016.

[9] J. Carletta, S. Ashby, S. Bourban, M. Flynn, M. Guillemot, T. Hain, J. Kadlec, V. Karaiskos, W. Kraaij, M. Kronenthal, G. Lathoud, M. Lincoln, A. Lisowska, I. McCowan, W. Post, D. Reidsma, & P. Wellner, “The AMI meeting corpus: A pre-announcement", *Proc. MLMI*, Bethesda, 2006.

[10] Z. Lin, M. Feng, C.N. dos Santos, M. Yu, B. Xiang, B. Zhou, & Y. Bengio, “A structured self-attentive sentence embedding", *Proc. ICLR*, Toulon, 2017.

[11] R. Lin, S. Liu, M. Yang, M. Li, M. Zhou, & S. Li, “Hierarchical Recurrent Neural Network for Document Modeling", *Proc. EMNLP*, Lisbon, 2015.

[12] R. Masumura, T. Tanaka, T. Moriya, Y. Shinohara, T. Oba, & Y. Aono, “Large Context End-to-end Automatic Speech Recognition via Extension of Hierarchical Recurrent Encoder-decoder Models", *Proc. ICASSP*, Brighton, 2019.

[13] G. Sun, C. Zhang, & P. C. Woodland “Speaker diarisation using 2D self-attentive combination of embeddings", *Proc. ICASSP*, Brighton, 2019.

[14] O. Chen, A. Ragni, M.J.F. Gales, & X. Chen “Active Memory Networks for Language Modeling", *Proc. ICASSP*, Calgary, 2018.

[15] Y. Wu, H. Yamamoto, X. Lu, S. Matsuda, C. Hori, & H. Kashioka. “Factored recurrent neural network language model in ted lecture transcription", *Proc. IWSLT*, pages 222–228, 2012.

[16] T. Mikolov & G. Zweig. “Context dependent recurrent neural network language model", *Proc. SLT*, pages 222–228, 2012.

[17] T.H. Wen, A. Heidel, H.Y. Lee, Y. Tsao, & L.S. Lee. “Recurrent neural network based personalized language modeling by social network crowd-sourcing", *Proc. Interspeech*, Florence, 2011.

[18] Y. Shi. *“Language models with meta-information."*, PhD thesis, Delft University of Technology, 2014.

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

[20] F. L. Kreyssig, C. Zhang, & P. C. Woodland *“Improved TDNNs using deep kernels and frequency-dependent grid-RNNs"*, *Proc. ICASSP*, Calgary, 2018.

[21] D. Povey, & P. C. Woodland *“Minimum phone error and I-smoothing for improved descriminative training."*, *Proc. ICASSP*, Orlando, 2002.

[22] C. Zhang, F. L. Kreyssig, Q. Li, & P. C. Woodland *“PyHTK: python library and ASR pipelines for HTK."*, *Proc. ICASSP*, Brighton, 2019.

[23] H. Xu, K. Li, Y. Wang, J. Wang, S. Kang, X. Chen, D. Povey, & S. Khudanpur *“Neural network language modeling with letter-based features and importance sampling"*, *Proc. ICASSP*, Calgary, 2018.

[24] J. Devlin, M. Chang, K. Lee, & K. Toutanova *“BERT: Pre-training of deep bidirectional transformers for language understanding"*, *arXiv:1810.04805.*

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, & I. Polosukhin *“Attention is all you need"*, *Proc. NIPS*, Long Beach, 2017.

[26] S. Katz *“Estimation of probabilities from sparse data for the language model component of a speech recognizer."*, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(3):400–401.

[27] R. Kneser & H. Ney *“Improved backing-off for n-gram language modeling"*, *Proc. ICASSP*, pages 181–184, 1995.

[28] S. Bengio, O. Vinyals, N. Jaitly, & N. Shazeer *“Scheduled sampling for sequence prediction with recurrent neural networks"*, *Proc. NIPS*, Montreal, 2015.

[29] R. Voleti, J. M. Liss, & V. Berisha *“Investigating the effects of word substitution errors on sentence embeddings"*, *Proc. ICASSP*, Brighton, 2019.

[30] X. Chen. *“Scalable recurrent neural network language models for speech recognition"*, PhD thesis, University of Cambridge, 2017.

[31] T. Mikolov & G. Zweig *“Context dependent recurrent neural network language model"*, *Proc. SLT*, Miami, 2012.

[32] P. Werbos *“Backpropagation through time: what it does and how to do it"*, *Proc. IEEE*, 78(10):1550–1560, 1990.

[33] J. Chung, C. Gulcehre, K. Cho & Y. Bengio *“Empirical evaluation of gated recurrent neural networks on sequence modeling"*, *Proc. NIPS*, Montreal, 2014.

[34] S. Hochreiter & J. Schmidhuber *“Long-short term memory"*, *Neural Computation* 9(8):1735 1780, 1997.

[35] T. Hoffman *“Probabilistic latent semantic analysis"*, *Proc. ACM SIGIR*, Berkeley, 1999.

[36] D. Blei, A. Ng & M. I. Jordan *“Latent dirichlet allocation"*, *Journal of Machine Learning Research*, pages 993-1022, 2003.

[37] Y. Teh, M. I. Jordan, M. Beal & D. Blei *“Hierarchical dirichlet processes"*, *Journal of the American Statistical Association*, pages 1566-1581, 2012.

[38] V. Peddinti, D. Povey, S. Khudanpur *“A time delay neural network architecture for efficient modeling of long temporal contexts"*, *Proc. Interspeech*, Dresden, 2015.

[39] W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang, & A. Stolcke *“The Microsoft 2017 conversational speech recognition system"*, *arXiv 1708.06073*, 2017.

Appendix A

Extra Experimental Results

Experiments on Eval Set

Experiments with Fixed Sentence Embeddings

RNNLMs	Substitution	Deletion	Insertion	Sentence Error	WER
Vanilla LSTM	13.0	7.0	3.5	54.8	23.5
Cross-utt. FC	12.9	7.0	3.5	54.6	**23.4**
Cross-utt. Atten.	12.9	7.1	3.6	54.6	23.5
Cross-utt. Hier.	13.0	7.2	3.7	56.2	23.9
CUED-RNNLM	13.2	7.3	3.7	56.9	24.3
Kaldi*	N/A	N/A	N/A	N/A	23.9

Table A.1 WER break-down comparison between different extractors on eval set. Acoustic model trained using MPE-criterion. FC represents the fully-connected layer for extractor. Atten. refers to self-attentive strucutre and Hier. refers to the hierachical LSTM structure.

Appendix B

Risk Assessment Retrospective

This project is completely computer-based. Therefore, potential hazards include arranging the working environment such as seating, monitor, keyboard suitably. The risk assessment form submitted in the Michaelmas term summarised the risks well which were encountered during the project (ergonomic issues). If I start the project again now, I will carry out the risk assessment in the same way I did in Michaelmas term.