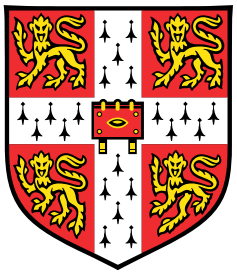
Uncertainty and confidence scores for sequence data



Alexandros Kastanos

Department of Engineering
University of Cambridge

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

Magdalene College August 2019

To my parents, for a quarter century of selfless dedication.
To Nicholas, for teaching me so much ...

Declaration

I, Alexandros Kastanos of Magdalene College, being a candidate for the MPhil in Machine Learning and Machine Intelligence, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Word count: 14732

Alexandros Kastanos
August 2019

Declaration

The core code which I received was written by Qiujia Li and Preben Ness. The existing LatticeRNN code, obtained from a Github repository, did not work at first. I made a pull request with the required fixes to the code. These modifications to the original code can be viewed at `https://github.com/qiujiali/lattice_rnn`.

I went on to extend the LatticeRNN code to support various attention mechanisms and bidirectional recurrent structures to include grapheme information. This work can be found at `https://github.com/alecokas/BiLatticeRNN-Confidence`.

Furthermore, I extended the existing lattice and confusion network processing code base, which was obtained from the CUED speech group machines, to store grapheme-marked structures in `*.npz` format. This work can be found in the Github repository located at `https://github.com/alecokas/BiLatticeRNN-data-processing`.

Together with other helper scripts which I created, this repository contains a tool for matching arcs in a confusion network to the corresponding arc in a lattice and extract additional information. The implementation and results chapters rely on the above mentioned software.

Alexandros Kastanos
August 2019

Acknowledgements

First and foremost, I would like to acknowledge my supervisor Professor Mark Gales, for in addition to setting up an interesting project, providing me with direction, technical support, and for taking extra time out to meet with me when requested. A big thank you to Dr Anton Ragni for always being available to provide advice and answer my questions whether via email or in person. Thank you to Jacob and Devin for your proofreading suggestions, and Antonio for your Pepys camaraderie. Finally, I would like to thank my college family, for being there in the early hours of the morning when it would have otherwise just been me and that duck quacking from the river Cam.

Abstract

In and of themselves, confidence scores are a useful indication of the robustness and reliability of the transcriptions produced by Automatic Speech Recognition (ASR) systems. The value of these confidence scores extends beyond the transcription to various downstream and upstream applications.

The work presented in this thesis begins with an overview of the ASR fundamentals and a focus on the different structures which can be obtained from the decoding process. The structures considered in this investigation are lattices, confusion networks, and one-best sequences. This leads to a discussion on traditional confidence estimation techniques such as word posterior probabilities. Following this, a discussion on deep learning techniques for sequence data, such as recurrent neural networks and attention mechanisms, is presented. Thereafter, end-to-end deep learning models, such as Bi-directional Long Short-Term Memory (BiLSTM) and LatticeRNN, are described for confidence score estimation. An extension of the LatticeRNN model to incorporate sub-word level features, such as graphemes, is proposed. This is accompanied by an algorithm for matching arcs between corresponding graph-like structures in order to obtain additional feature estimates. Empirical experimentation quantitatively demonstrates that the introduction of grapheme features can lead to improved confidence scores. Finally, a discussion of the results and suggestions for future work are presented.

Table of contents

List of figures

List of tables

Nomenclature

Acronyms / Abbreviations

AM Acoustic Model

ASR Automatic Speech Recognition

AUC Area Under the Curve

CBOW Continuous Bag of Words

CPU Central Processing Unit

CRF Conditional Random Fields

CUED Cambridge University Engineering Department

DAG Directed Acyclical Graph

FPR False Positive Rate

GPU Graphics Processing Unit

GRU Gated Recurrent Unit

GSF Grammar Scale Factor

HMM Hidden Markov Model

LM Language Model

LSTM Long Short Term Memory

MAP Maximum A Posteriori

MBR Minimum Bayes Risk

NCE Normalised Cross Entropy

RNN Recurrent Neural Network

ROC Receiver Operating Characteristics

SGD Stochastic Gradient Descent

t-SNE t-Distributed Stochastic Neighbour Embedding

TBPTT Truncated Backpropagation Through Time

TPR True Positive Rate

UAPS Unique Arcs Per Second

WER Word Error Rate

WTS Word Trellis Stability

Chapter 1

Introduction

This chapter opens the discussion on uncertainty and confidence scores for sequence data. First, the motivation for the work is presented. This is followed by outlining the aims and major contributions before presenting the structure of the remaining chapters in the document.

1.1 Motivation

The continued integration of speech enabled solutions into everyday devices has elevated interest in Automatic Speech Recognition (ASR) systems. Naturally, this leads to heightened expectations in terms of performance and functionality. This is evident from the increased popularity of voice activated personal assistants, such as Amazon's Alexa and Apple's Siri, as well as increased usage of ASR in broadcast media. Fortunately, the ongoing developments in deep learning has seen the state of the art in ASR accuracy steadily improve in recent years, thereby maintaining the relevance and usefulness of these technologies. Fundamentally, ASR systems are predictive tools which produce a number of ranked hypothesis transcriptions. A measure of certainty in the prediction is a useful feature to use internally for accuracy improvement as well as for providing user feedback on the reliability of any candidate transcription.

Beyond the simple case of user feedback, confidence scores as a certainty metric are useful to a number of upstream and downstream applications. In machine translation the ability to send a low confidence warning to indicate a high level of uncertainty between a number of hypotheses has a powerful impact on downstream decision making. Similarly, providing upstream applications, such as speaker adaptation and semi-supervised training, with access to confidence estimates improves the reliability of the ASR for these applications [2, 3]. The value of confidence metrics is exaggerated for low resource languages where the Word Error Rate (WER), and thus the proportion of incorrect hypotheses, is significantly

higher than languages such as English and Spanish. For this reason, the Cambridge University Department of Engineering (CUED) Georgian language ASR system is used as a test bench for the confidence score investigation.

1.2 Aims and Contributions

This work looks to improve on existing deep learning techniques for word-based confidence score estimation. In particular, a framework which scales for confidence estimates beyond the simple one-best sequence to lattice and confusion network structures is desired. The model presented expands on the existing LatticeRNN model for confidence score estimation by integrating grapheme level features. To this end, a grapheme encoder for reducing the grapheme feature sequence into a fixed form representation as well as the development of grapheme embeddings for the Georgian language is contributed. Additionally, an arc matching technique for incorporating approximate features into confusion networks and one-best sequences from a corresponding lattice is developed [1].

1.3 Overview of Chapters

The remaining chapters in this thesis are organised as follows:

- **Chapter 2** serves as an introduction to Automatic Speech Recognition with a focus on the graph-like structures attainable from the decoding process.
- **Chapter 3** presents background on the topic of confidence scores. More traditional techniques for estimating confidence are discussed along with some of the applications for confidence estimation.
- **Chapter 4** introduces deep learning for sequence data. This provides the reader with technical information on recurrent neural network models and attention techniques which make handling sequence data in an end-to-end deep learning framework possible.
- **Chapter 5** leverages the previous two chapters to present a technique for incorporating deep learning for confidence score estimation. Specifically, LatticeRNN for confidence scores and the improvements proposed in this work.
- **Chapter 6** deals with practical implementation details such as the data processing pipeline.

[1] The source code is available at: `https://github.com/alecokas/BiLatticeRNN-Confidence`

- **Chapter 7** presents the quantitative analysis of the proposed confidence score prediction model. The experiments are supplemented with a discussion of the consequences of the empirical results and a description of the experimental setup.
- **Chapter 8** concludes this work with a summary of the key discussion points and suggests improvements for future development.

Chapter 2

Automatic Speech Recognition

The aim of an ASR system is to generate the correct transcription for a given audio sequence. Specifically, this requires that the audio sequence is mapped to the corresponding word sequence where the audio sequence is defined as an observable sequence of utterances, $\mathbf{o}$, and the word sequence is simply described by $\mathbf{w}$. A high level overview of a traditional ASR system is given in figure 2.1, where each of the key components are described in the sections to follow.

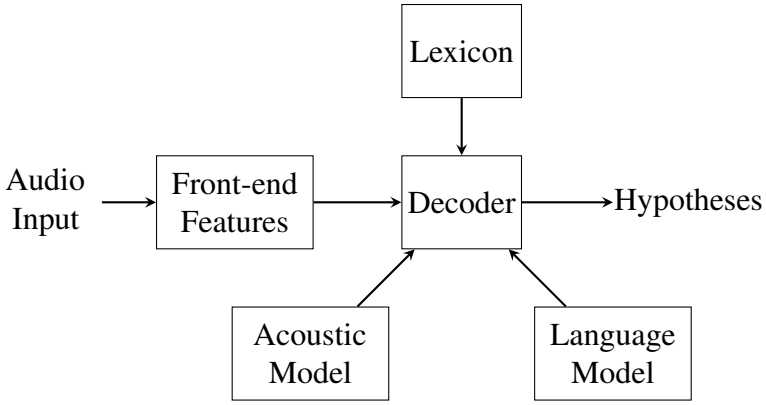


Fig. 2.1 Overview of an ASR system

A mathematical model for describing the ASR system can be defined by the process which aims to maximise the posterior probability, $P(\mathbf{w}|\mathbf{o})$, of the word sequence $\mathbf{w}$ given the acoustic sequence $\mathbf{o}$. This is an application of Bayes' decision rule where the maximum a posteriori (MAP) estimate is used to estimate the most probable word sequence, $\hat{\mathbf{w}}$, given the utterance [4].

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} P(\mathbf{w}|\mathbf{o}) = \arg\max_{\mathbf{w}} \frac{p(\mathbf{o}|\mathbf{w})P(\mathbf{w})}{p(\mathbf{o})} \tag{2.1}$$

Since $p(\mathbf{o})$ does not have any effect on the optimisation, it can be dropped from the expression. Hence, the model is simplified to the expression in equation 2.2. The two remaining terms in the expression, $p(\mathbf{o}|\mathbf{w})$ and $p(\mathbf{w})$ are given by the acoustic and language model scores respectively. The language model provides the prior over the word sequence while the acoustic likelihood given the word sequence is determined by the acoustic model.

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} p(\mathbf{o}|\mathbf{w})P(\mathbf{w}) \tag{2.2}$$

2.1 Acoustic Models

As mentioned above, the acoustic model is responsible for providing a conditional likelihood score for the observed utterances given a word sequence. A popular choice for this acoustic model is a Hidden Markov Model (HMM) where a distinct HMM is trained for a sub-word unit such as a phoneme or grapheme. Due to the non-emitting entry and exit states, sub-word HMMs can be combined through composition to build word models from the sub-word level. The motivation for training a HMM for each sub-word rather than the word-level derives from the large vocabulary size requirements of ASR, together with Zipfth's law for word frequency which states that too many words occur too infrequently to train a distinct HMM for each word.

Figure 2.2 provides a graphical model representation of a HMM where the state transition probability between hidden states q_t and q_{t+1} is given by $a_{q_t,q_{t+1}}$. The emission probability of state q_t producing observation vector $\mathbf{o}_t$ is given by $b_{q_t}(\mathbf{o}_t)$. The observation vectors are presented in shaded nodes to indicate that these are observed variables.

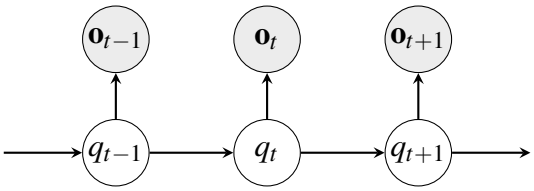


Fig. 2.2 Graphical model representation of a HMM

HMMs obey the first-order Markov assumption and assume conditional independence between hidden states. These two assumptions are defined in equations 2.3 and 2.4 respectively.

$$P(q_{t+1}|q_0, q_1, \ldots, q_t) = P(q_{t+1}|q_t) = a_{q_t,q_{t+1}} \tag{2.3}$$

$$p\left(\mathbf{o}_t|q_0,q_1,\ldots,q_t\right)=p\left(\mathbf{o}_t|q_t\right)=b_{q_t}\left(\mathbf{o}_t\right) \tag{2.4}$$

As a result, the conditional likelihood of an observation sequence is obtained by marginalising over all possible hidden state sequences as shown below

$$p(\mathbf{O}|\boldsymbol{a})=\sum_{\mathbf{q}\in\mathbf{Q}}p(\mathbf{O},\mathbf{q}|\mathbf{a})=\sum_{\mathbf{q}\in\mathbf{Q}}\prod_{t=0}^{T-1}a_{q_t,q_{t+1}}b_{q_t}\left(\mathbf{o}_t\right) \tag{2.5}$$

where $Q=\{q_0,\ q_1,\ldots,q_t\}$ and $\mathbf{O}=\{\mathbf{o}_0,\ \mathbf{o}_1,\ldots,\mathbf{o}_t\}$. A large corpus of labelled training data is required to train such a model. This results in long training times with a single epoch of training possibly taking several hours [5]. Recent advancements in acoustic models include a recurrent neural network architecture called Grid-RNN which has been shown to reduce error rates in ASR systems [6].

In this work, a graphemic ASR system for the Georgian language, as presented in section 7.1.1, is used as the platform from which to run experiments. Hence, the acoustic model in this particular application operates on the grapheme unit level. In order to map between the acoustic level representation and the word sequence, **w**, a grapheme lexicon is required [7].

2.2 The Lexicon

A lexicon is required to map between words and their corresponding grapheme sequences, where a grapheme is defined as the smallest written unit in a language [8]. For example, in the English language, the word `fox` is comprised of the grapheme sequence $\{\texttt{\textbackslash f},\texttt{\textbackslash o},\texttt{\textbackslash x}\}$ where the alphabetical characters are the graphemes. The lexicon defines a vocabulary of words which can be described as being 'known' to the system as well as mapping each lexical item to one or more pronunciations. These two features allow the lexicon to achieve its overall objective. An advantage of using a grapheme-based system over a phoneme-based system, is that the spelling of the words can be used directly to obtain the graphemes without the need for the lexicon to be derived from extensive expert knowledge [9]. For low-resource languages, such as Georgian, this kind of expert knowledge is particularly hard to come by, hence a graphemic system provides a clear advantage in terms of obtaining a lexical model.

2.3 Language Models

The prior in equation 2.2 is determined by a language model. For a word sequence **W** of length T, the language score is defined as the joint probability of each word as provided in equation 2.6 where w_0 is the start of sentence token.

$$P(\mathbf{W}) = P(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T+1} P(w_t | w_0, \ldots, w_{t-2}, w_{t-1}) \tag{2.6}$$

Once again, the large vocabulary size of ASR means that determining this joint probability is computationally expensive to the point of being infeasible without introducing a simplifying assumption. To address the problem of intractability, the concept of N-grams is introduced where it is assumed that histories are equivalent if the most recent N-1 words are the same. The formulation above can be defined as a 1-gram (unigram) model under this framework. For the generalised N-gram, the joint probability of the word sequence **W** is estimated using equation 2.7.

$$P(\mathbf{W}) \approx \prod_{t=1}^{T+1} P(w_t | w_{t-N+1}, \ldots, w_{t-1}) \tag{2.7}$$

As N is increased, the more accurate a prediction the system can make, however the effective vocabulary size increases exponentially. If V is the number of words in the vocabulary, the number of N-grams in the vocabulary is defined by V^N. Because of the finite amount of training data available, a compromise between the ability to train such an N-gram system and the rigidity of the N-gram assumption is required. Techniques such as discounting, which adds sudo-counts to empirically improbable events, and backing-off to smaller N-grams for rare N-gram combinations can be used to improve language model performance [10].

2.4 Decoding

Typically the output from an ASR is a set of N-best hypotheses. This can be efficiently represented in the form of a word lattice which contains a large number of candidate hypothesis transcriptions for a given speech segment. The likelihood of each hypothesis is used as a metric for determining the probability of each hypothesis being correct [11]. The task of the decoding process is to search for the word sequence **w** which maximises equation 2.2, thereby determining the one-best path through the lattice.

The subsections to follow discuss the three types of structures which will be dealt with in this work: lattices, confusion networks, and one-best sequences. Figure 2.3 visually introduces these topologies.

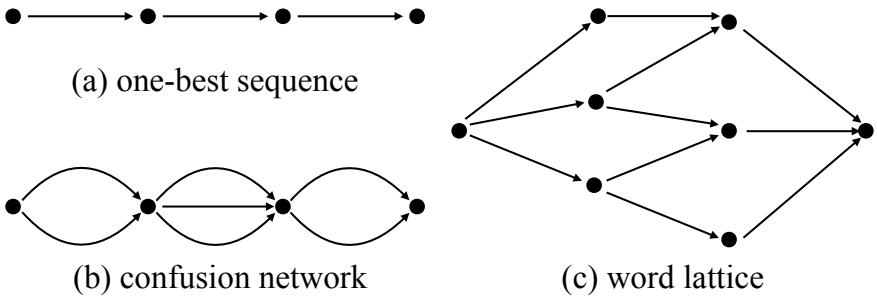


Fig. 2.3 A simple illustration of each of the three graph-like structures considered in this work [1].

2.4.1 Lattices

A lattice is a particular form of Directed Acyclc Graph (DAG) where a left-to-right information flow is typically used to mirror the left-to-right movement of written text. A word lattice is made up of a set of nodes, $\mathscr{N}$, and arcs, $\mathscr{E}$, with a single word and its corresponding information being represented on a single arc. Figure 2.4 is a contrived example of a lattice output for the phrase 'quick brown fox'.

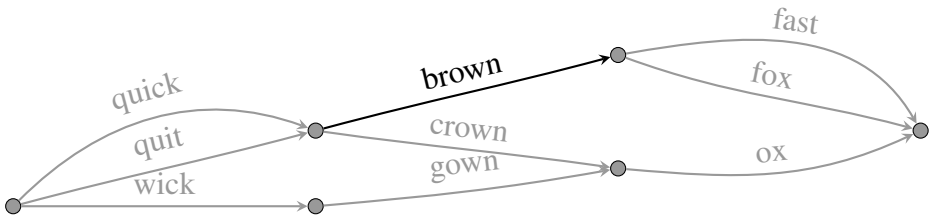


Fig. 2.4 A simple word-marked lattice for the example utterance: 'quick brown fox'.

A close up inspection of the arc which represents the word `brown`, as shown in figure 2.5, provides an opportunity to describe some of the key lattice elements for this report. Two distinct sets of arcs and nodes can be defined. One of these comprises of all ancestors $\left(\vec{\mathscr{N}}, \vec{\mathscr{E}}\right)$ while the second is made up of all descendants $\left(\overleftarrow{\mathscr{N}}, \overleftarrow{\mathscr{E}}\right)$. The start and end times of edge e_i are denoted by t_i^s and t_i^f respectively. Since only the start time and duration of an arc is stored on each arc, the end time is calculated in an online fashion as required through simple addition as shown below.

$$t_i^f = t_i^s + t_i^d \tag{2.8}$$

Furthermore, each arc contains the language model score, LM_i, and acoustic model score AM_i for the word $\mathbf{w}_i$ described by the arc. In order to insert the word itself into the lattice framework, a continuous variable vector representation of each word in the vocabulary is required. This is commonly done via a word embedding process, which is discussed in greater detail in chapter 5.

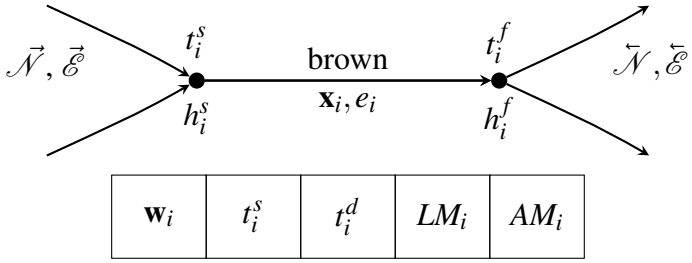


Fig. 2.5 The edge for the word 'brown' isolated from the lattice.

Sub-word level information can be added to give the lattice an additional layer of context and richness. As already mentioned, this work uses a graphemic ASR system, thus the arcs of the lattice can be grapheme-marked via determinization to allow grapheme level features, such as duration, to be obtained.

Lattice Pruning Strategies

Due to the vast nature of the lattices produced by modern ASR systems, there is a need to reduce the computational complexity of the decoding process. This is achieved through a task called pruning that aims to only maintain the paths through the lattice which have a significant chance of being the correct transcription.

One pruning strategy involves using a token passing algorithm which keeps track of the partial information along each path at each step along the sequence. At each time step, t, all paths which have a partial log probability less than a beam-width threshold below the path with the highest log probability are discarded. This method is called beam pruning [5].

An alternative strategy is unique arcs per second (UAPS) pruning which selects the arcs to prune at each time instance such that a pre-defined distribution over words is maintained. The aim of this method is to reduce the number of identical words which only differ slightly in terms of start time, end time, and likelihood. A comparative study indicates that UAPS pruning is able to maintain a similar number of unique arcs to the original lattice, whilst significantly reducing the total number of arcs in the lattice. This is an advantage of UAPS

lattices since it results in a better balance between maintaining a diverse set of competing paths and still providing a computational boost by reducing the total number of arcs in the lattice [9].

2.4.2 Confusion Networks

Confusion networks are an alternative representation of a lattice in which all paths through the linear graph pass through all nodes in the network. This is achieved by two consecutive arc clustering processes whereby overlapping arcs are merged such that the probability distribution over all competing arcs sums to one. Although it is possible to merge arc posteriors, merging the language and acoustic model scores is meaningless. As a result, confusion networks generated from lattices are limited to the word, start time, duration, and arc posterior information. The confusion network generation process is an example of Minimum Bayes Risk (MBR) decoding whereby the word error is minimised. This contrasts with the Viterbi decoding process which operates at a sentence level [12]. An advantage of confusion networks is that since all paths must pass through each node, the sum of arc posteriors for each node must be less than or equal to 1. This observation allows an explicit estimation of the probability of deletion to be given by the probability mass associated with the null arc [13].

2.4.3 One-Best Sequences

There are two ways to obtain the one-best sequence from an ASR system, either from the lattice directly or from the confusion network. When deriving the one-best from a lattice, the Viterbi algorithm is used to obtain the most likely sequence of arc posteriors. In this scenario, the one-best sequence has access to all the same arc information as the lattice. Since confusion networks are derived from MBR decoding, extracting the one-best sequence from a confusion network is as simple as selecting the maximum arc posterior between each pair of sequential nodes, called a slot. As a result, obtaining the one-best sequence from a confusion network limits the one-best sequence to the information already existing in the confusion network.

2.5 Evaluating ASR predictions

In order to determine the accuracy of an ASR system, a robust metric for evaluating each hypothesis transcription is required. The most commonly used metric is called Word Error Rate (WER)

$$\text{WER} = \frac{\text{N(Ins)} + \text{N(Del)} + \text{N(Sub)}}{\text{N(Words in Reference)}} \tag{2.9}$$

where N(Ins), N(Del), and N(Sub) are the number of insertion, deletion, and substitution errors respectively. These are the three types of errors associated with ASR systems. An insertion error occurs when an extra word is inserted into the hypothesis sequence while a deletion error occurs when the hypothesised sequence does not contain a prediction for a given reference word. Substitution errors are recorded when the ASR produces an incorrect prediction for a reference word. The numerator of the above equation is referred to as the word error, or edit distance, between the hypothesis and reference words. The Levenshtein distance between two words w_1 and w_2, as defined in equation 2.10, is a common mathematical formulation for determining the edit distance where L_1 and L_2 are the character lengths of the two respective words.

$$\mathscr{L}_{w_1,w_2}(L_1,L_2) = \begin{cases} \max(L_1,L_2) & \text{if } \min(L_1,L_2) = 0 \\ \min \begin{cases} \mathscr{L}_{a,b}(L_1-1,L_2)+1 & \text{(Deletion)} \\ \mathscr{L}_{a,b}(L_1,L_2-1)+1 & \text{(Insertion)} \\ \mathscr{L}_{a,b}(L_1-1,L_2-1)+\mathbf{1}_{(a_{L_1}\neq b_{L_2})} & \text{(Substitution)} \end{cases} \end{cases} \tag{2.10}$$

Chapter 3

Confidence Scores

In chapter 2, the generation of hypothesis ASR transcriptions was presented. A useful attribute for an ASR system is to provide an indication of whether the hypothesised transcriptions are correct or incorrect. This chapter discusses confidences scores as an indicator for this exact purpose.

3.1 Confidence Scores for ASR

A challenge to consider when setting up a mathematical framework from which to evaluate uncertainty is the ambiguity and abstract nature of assigning a confidence score to a prediction. There are a number of abstraction levels at which confidence can be considered such as the word level, utterance level, and concept level [14]. One way to approach this problem is to estimate the probability of a particular word in a transcription being correct using predictive features from the underlying ASR system [15]. Alternatively, a measure of confidence can be constructed as the posterior probability of a particular arc taken directly from the ASR lattice or N-best list [16]. The second approach is adopted as the starting point for this work where confidence is assigned on a per-word basis.

Figure 3.1 is an example of what a word-based confidence score result may look like for a one-best sequence prediction of a simple utterance waveform. The one-best word and alignment predictions are supplied below the respective audio segments in figure 3.1. Conventional confidence score estimation methods only focus on providing estimates for hypothesised words. Deleted words, as demonstrated by the missing prediction for the word *fox*, are often neglected as there is no hypothesised word for which to produce a prediction. This is problematic because not only do deletion errors have a severe impact on upstream and downstream applications, but deletion error rates tend to significantly increase when moving between deployment domains.

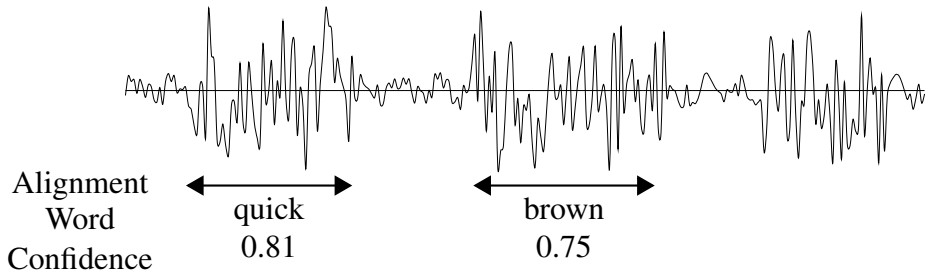


Fig. 3.1 Example ASR transcription and confidence estimation of the phrase *quick brown fox* where the word *fox* has been deleted.

3.2 Arc Posterior Probabilities

For word-based lattices, the arc posterior probability $p(e|\mathbf{O},\lambda)$ is identical to the word posterior probability $p(w|\mathbf{O},\lambda)$. Given that each arc in the lattice contains an acoustic model score and a language model score as described in the previous chapter, the joint probability for a given path through the lattice, $\mathbf{q}$, and an utterance $\mathbf{O}$ can be defined by equation 3.1. The independence assumptions introduced when deriving this model result in the emission probabilities being underestimated. To counteract this undesirable behaviour without altering the one-best sequence, the acoustic score is reduced using the grammar scale factor (GSF), γ.

$$p(\mathbf{q},\mathbf{O}) = p(\mathbf{O}|\mathbf{q})^{1/\gamma} P(\mathbf{W}) \tag{3.1}$$

From this joint probability, the forward-backward algorithm can be used to efficiently determine the arc posterior probability in a lattice, which forms the basis of an estimate for confidence.

3.2.1 Forward-Backward for Lattice Arc Posteriors

The forward-backward algorithm is typically used for inference using HMMs by determining the probability of a state sequence. In the context of a lattice, the problem is framed in terms of determining the arc posterior probability, $p(e|\mathbf{O},\lambda)$, rather than the state posterior probability, $p(\zeta_t|\mathbf{O},\lambda)$. By representing the arc posterior through Bayes' theorem, it becomes clear that a naive implementation requires summing over all paths in the lattice which pass through arc e.

$$p(e|\mathbf{O},\lambda) = \frac{\sum_{\mathbf{q}\in Q_e} p(\mathbf{q},\mathbf{O}|\lambda)}{p(\mathbf{O}|\lambda)} \quad \text{where} \quad p(\mathbf{O}|\lambda) \approx \sum_{\mathbf{q}\in Q} p(\mathbf{q},\mathbf{O}|\lambda) \tag{3.2}$$

Q_e is the set of all paths q which pass through the arc e for the utterance $\mathbf{O}$, Q is the set of all paths in the lattice, and λ is the model. This regime is too computationally expensive to be a practical solution, thus the forward-backward algorithm is used. As the suggests, the forward-backward algorithm is made up of two distinct calculations.

Forward Algorithm

The forward probability computes the likelihood of the total observation sequence, $p(\mathbf{O}|\lambda)$ and is formulated in the three steps given below [17].

1. Initialisation of the forward log probability, α_{root}, at the root node, e_{root}:

$$\alpha_{root} = \texttt{minimum} \tag{3.3}$$

2. Forward calculation of the joint log probability for each arc in $\mathscr{E}$:

$$\alpha_i^s = \log\Big(\sum_j \exp(\alpha_j)\Big) \quad \text{where } j \text{ indexes all incoming arcs to } e_i \tag{3.4}$$

$$\alpha_i = \alpha_i^s + \frac{1}{\gamma}\log(\mathrm{AM}_i) + \log(\mathrm{LM}_i) \tag{3.5}$$

3. Termination at the end node to obtain the total likelihood:

$$\log(p(\mathbf{O}|\lambda)) = \alpha_{end} \tag{3.6}$$

Similarly, the likelihood of the observation sequence can be determined using the backward algorithm by propagating information from the terminal node, through to the root node.

Backward Algorithm

1. Initialisation of the backward log probability for the end node, β_{end}:

$$\beta_{end} = \texttt{minimum} \tag{3.7}$$

2. Backward calculation of the conditional log probability for each arc e_i in $\mathscr{E}$:

$$\beta_i^f = \log\left(\sum_j \exp\left(\alpha_i + \frac{1}{\gamma}\log\left(\mathrm{AM}_i\right) + \log\left(\mathrm{LM}_i\right)\right)\right)$$

where j indexes all outgoing arcs from e_i (3.8)

$$\beta_i = \beta_i^f \tag{3.9}$$

3. Termination at the end node to obtain the total likelihood:

$$\log\left(p(\mathbf{O}|\lambda)\right) = \beta_{root} \tag{3.10}$$

The forward-backward algorithm is most useful when considering how to determine the posterior probability for a particular arc in the lattice, $p(e|\mathbf{O})$. Due to the asymmetric nature of the forward and backward passes, the terminating conditions are modified such that the forward algorithm terminates at the start of the arc e while the backward algorithm terminates at the end of arc e. Hence the lattice arc posterior is given by equation 3.11, where $Z = \sum_{e' \in \varepsilon} \exp\left(\alpha_{e'} + \beta_{e'}\right)$ is introduced as a normalising constant to ensure that $p(e|\mathbf{O})$ is a valid probability [1].

$$p(e|\mathbf{O}) = \exp\left(\alpha_e + \beta_e - \log Z\right) \tag{3.11}$$

3.2.2 Decision Trees For Mapped Posteriors

Decision trees are a commonly used technique for confidence score estimation which allow the posterior probability of a word being correct given the corresponding feature vector to be estimated directly. This circumvents the need to generate the MAP estimate, $p(\bar{c}_t = 1|x_t)$, via the conditional probabilities using Bayes theorem as shown below.

$$c_t = p(\bar{c}_t = 1|x_t) = \frac{p(\bar{c}_t = 1)p(x_t|\bar{c}_t = 1)}{p(x_t|\bar{c}_t = 1) + p(x_t|\bar{c}_t = 0)} \tag{3.12}$$

The binary tree is developed through a series of uni-variate conditionals operating on the feature vector to form a piece-wise monotonically increasing mapping which compensates for the overestimation of the predicted confidence score given by the arc posterior probabilities [18]. A decision tree made up of 8 leaf nodes was used in [19] as part of a system which outperformed contemporary confidence score estimation techniques. The gains of decision tree based systems are obtained by reducing the overconfident confidences scores which

emerge from arc posteriors estimates. Figure 3.2 is an example of the piece-wise mapping typically generated by a decision tree compared to the unmapped probabilities.

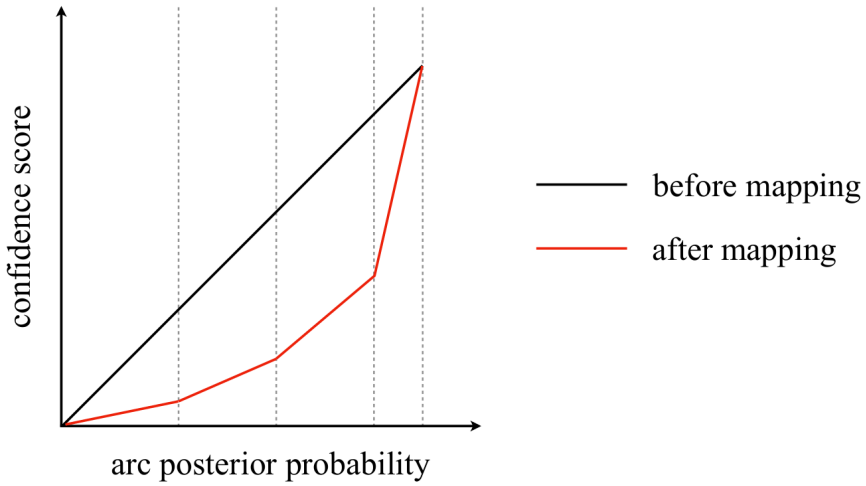


Fig. 3.2 Piece-wise mapping produced by applying the decision tree [1].

3.3 Features for Confidence Estimation

Rather than simply using the posterior probability as the only feature for confidence estimation, one may consider extracting additional features at the decoding stage. By selecting a complimentary set of features, the overall accuracy of the confidence estimation model is expected to increase. An overview of candidate features obtainable from the decoding process are introduced below.

Acoustic Model Features

The normalised acoustic likelihood, as described in section 2.1, for a sub-word or a full word, is derived by taking the geometric mean of the acoustic likelihood of each individual state [20]. Acoustic stability as a feature for confidence is based on the premise that fluctuations in GSF should present larger variability in the decoded result for erroneous ASR outputs than error free outputs. This is equivalent to perturbing the ASR model and measuring the robustness of the system to noise. Higher acoustic stability is an indicator that greater confidence should be associated with a prediction. This idea leads naturally to viewing confidence estimation from a perturbation analysis perspective [21]. These two acoustic features can be generated for the sub-word level, thereby providing the option of additional sub-word level features [22].

Language Model Features

The language model score itself can be used as a feature where the greater the joint probability over the word sequence, $P(\mathbf{W})$, the more confidence one expects to be assigned to the prediction. As introduced in section 2.3, the back-off behaviour is an indicator of the simplifying assumptions applied by the language model. This reduction of the scope of the dependence assumption in the word sequence is expected to be an indicator of uncertainty [23].

Lattice-based Features

By leveraging the knowledge that a transcribed word is more likely to be correct if it occupies the same start and end times in a large proportion of competing lattice paths and is insensitive to a variations in the GSF, the Word Trellis Stability (WTS) feature is defined. This feature leverages the N competing hypotheses, often presented in the form of a lattice or confusion network, to provide an indication of arc confidence [24]. Another lattice-based feature, hypothesis density, is a metric for the number of competing arcs across a particular time interval. The more competing arcs over a given time interval, the higher the hypothesis density, and the lower the expected confidence score. As described in section 2.4, lattices are often pruned to remove unlikely arcs. The pruning strategy as well as the amount of pruning applied has a direct impact on the hypothesis density [25].

Duration Features

The duration at an HMM state, sub-word, and word level is a candidate feature for predicting confidence. It is hypothesised that a shorter duration is indicative of higher uncertainty. Empirical experimentation in [26] indicates that word duration has value as a feature for word level confidence estimation.

Utterance Features

A multitude of utterance level features for confidence estimation are described in [27]. In addition to being used as features in their own right, they can be used to generate an utterance feature score which is useful as a word level feature. Some of these utterance features are lexical score drop and utterance length.

3.4 Conditional Random Fields

Conditional Random Fields (CRFs), which were first proposed for segmenting and labelling sequence data, have since been used extensively to combine various features to improve confidence score estimations [28]. Given a set of model parameters, $\theta = (\lambda_1, \ldots, \lambda_K, \mu_1, \ldots, \mu_J)$, which are estimated through maximum likelihood training and gradient descent based optimisation techniques, the sequence of confidence scores can be modelled by equation 3.13.

$$p_\theta(\mathbf{c}|\mathbf{x}) \propto \exp\left(\sum_k \lambda_k a_k(c_k, \mathbf{x}) + \sum_j \mu_j g_j(\mathbf{c}, \mathbf{x})\right) \tag{3.13}$$

where $a_k(c_k, \mathbf{x}_t)$ are the transmission features and $g_j(\mathbf{c}, \mathbf{x})$ are the emission features. This is the form of a linear chain CRF which typically uses a discrete representation for the hidden variable $\mathbf{c}$. This can be achieved by defining the transmission function $g(\cdot)$ using quantisation with binning or spline features and distribution constraints [29, 15]. The graphical model for the linear chain CRF, shown in figure 3.3, indicates how the CRF is related to an HMM. The major difference between these two models is that the CRF is undirected, while HMMs are directed.

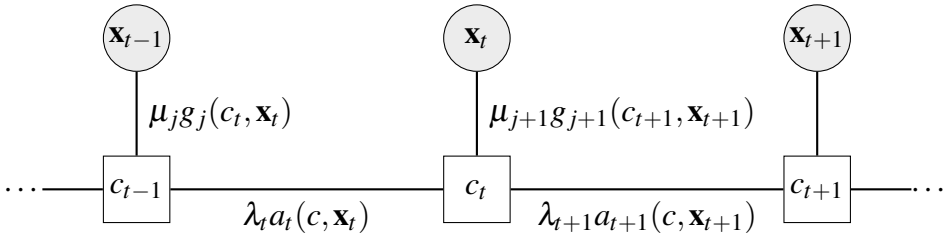


Fig. 3.3 Graphical model of a linear chain CRF.

One of the main advantages of CRFs for confidence scores, is that they allow variable sized feature vectors to be used as inputs to the model. This means that rather than depending on a single feature, as the decision tree mapped confidence estimates do, the combined discriminative power of a number of features can be leveraged. The sequential nature of CRFs has allowed previous work to explicitly predict deletion errors on one-best sequences [30]. This approach was extended in [31] to use a bi-directional recurrent neural network to predict confidence and deletion estimates explicitly.

Chapter 4

Deep Learning for Sequence Data

This chapter explores deep learning techniques for two particular sequence modelling problems. The first of these are sequence-to-sequence model where both sequence are of the same length. Although transformer style models have recently produced state of the art results in this field, the discussion is focused on recurrent methods [32, 33]. The second problem area is predicting a single value which compresses a variable length sequence. For this discussion, various forms of attention are presented.

4.1 Recurrent Neural Networks

Traditional neural networks, although incredibly useful for learning hierarchical representations, are restricted by the assumption of independence between samples used during the training and validation procedure. Recurrent Neural Networks (RNNs) are a powerful tool which overcome this limitation by propagating information forward through the sequence in the form of a hidden state vector. The ability to incorporate the new information from each sample into this hidden state vector is what allows RNNs to learn long range dependencies. A significant advantage of RNNs over Markov models is that they do not suffer from the same degradation in inference speed and expansion of the state-transition table as the state space increases [34].

4.1.1 Uni-Directional Recurrent Neural Networks

A uni-directional RNN generates the hidden representation at time t, $\mathbf{h}_t$, by applying an element-wise non-linearity, ϕ, to the result of the addition of the linearly transformed input vector at time t and the linearly transformed hidden state at time $t-1$. This propagation of

information is defined in equation 4.1, where $\mathbf{W}_f$ and $\mathbf{W}_h$ are matrices made up of trainable weights.

$$\mathbf{h}_t = \phi\left(\mathbf{W}_f^T \mathbf{x}_t + \mathbf{W}_h^T \mathbf{h}_{t-1}\right) \tag{4.1}$$

In a supervised setting, the predictions, y_t, are generated by a function $f_y(\cdot)$ which maps to the output space.

$$y_t = f_y\left(\mathbf{h}_t\right) \tag{4.2}$$

Recurrent structures are often visualised by unfolding the network over time as presented in figure 4.1. This provides an illustration of how the backpropagation algorithm can be trained across multiple time steps. However, as the sequence grows large, the problem of vanishing and exploding gradients hinders training [35]. A common remedy for this is truncated backpropagation through time (TBPTT), which limits the number of time steps which the error is propagated [36], thereby halting the vanishing or exploding phenomenon.

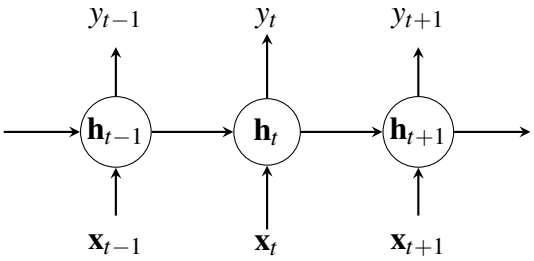


Fig. 4.1 Unfolded RNN structure

4.1.2 Bi-Directional Recurrent Neural Networks

The uni-directional recurrent structure described above only makes use of the current input and the history of preceding inputs to predict the current output. In many applications, such as natural language processing and speech, the current output is often dependent on a future input. For offline applications where the full sequence is available at the time of prediction a Bi-directional Recurrent Neural Network (BiRNN) can be used to incorporate past inputs as well as future inputs. This is achieved by two separate sets of hidden neurons where the first set is responsible for propagating information forward in time, while the second set propagates information backwards in time [37]. The hidden vectors for the forward and backward directions are denoted by $\overrightarrow{\mathbf{h}}_t$ and $\overleftarrow{\mathbf{h}}_t$, and defined in equations 4.3 and 4.4 respectively. The weight matrices $\mathbf{W}_x$, $\mathbf{W}_f$, and $\mathbf{W}_b$ which linearly transform the input vector

and the previous state vectors for the past and future information are trained similarly to the vanilla RNN.

$$\overrightarrow{\mathbf{h}}_t = \phi\left(\mathbf{W}_x^T \mathbf{x}_t + \mathbf{W}_f^T \overrightarrow{\mathbf{h}}_{t-1}\right) \tag{4.3}$$

$$\overleftarrow{\mathbf{h}}_t = \phi\left(\mathbf{W}_x^T \mathbf{x}_t + \mathbf{W}_b^T \overleftarrow{\mathbf{h}}_{t+1}\right) \tag{4.4}$$

The prediction for time step t is generated by applying the function $f_y(\cdot)$ to the concatenation of the forward and backward hidden representations at time t.

$$\mathbf{h}_t = \begin{bmatrix} \overrightarrow{\mathbf{h}}_t & \overleftarrow{\mathbf{h}}_t \end{bmatrix}^\top \tag{4.5}$$

$$y_t = f_y(\mathbf{h}_t) \tag{4.6}$$

The relationship between the input sequence, bidirectional hidden sequences, and the final prediction is schematically illustrated by figure 4.2.

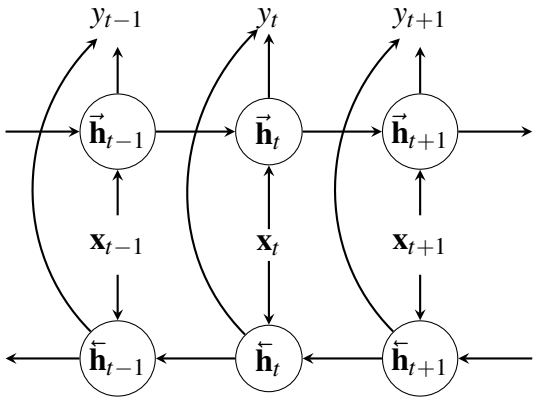


Fig. 4.2 Unfolded BiRNN structure

4.1.3 Long Short-Term Memory

Long Short-Term Memory (LSTM) units are a type of recurrent layer which contain an explicit memory cell for the purpose of enhanced long range context [38, 39]. The contents of the memory cell are controlled by two gating mechanisms called the input gate and the forget gate. Their respective functions are defined in equations 4.7 and 4.8, where $\sigma(\cdot)$ represents the sigmoid function, $\mathbf{W}_i^x$, $\mathbf{W}_i^h$, $\mathbf{W}_f^x$, and $\mathbf{W}_f^h$ are trainable weight matrices, and $\mathbf{b}_i$ and $\mathbf{b}_f$ are biases. The input gate influences the degree to which the current input contributes

to the memory cell at any given time t. In a similar fashion, the forget gate impacts the contribution of the previous state of the cell.

$$\mathbf{i}_t = \sigma\left(\mathbf{W}_i^x \mathbf{x}_t + \mathbf{W}_i^h \mathbf{h}_{t-1} + \mathbf{b}_i\right) \tag{4.7}$$

$$\mathbf{f}_t = \sigma\left(\mathbf{W}_f^x \mathbf{x}_t + \mathbf{W}_f^h \mathbf{h}_{t-1} + \mathbf{b}_f\right) \tag{4.8}$$

Equation 4.9 demonstrates a traditional configuration for using these two gating functions alongside a $\tanh(\cdot)$ activation to ensure that the memory cell contents are shielded from irrelevant inputs. The memory cell at time t is represented by $\mathbf{ç}_t$ and the weight matrices and biases are defined as described those described for the input and forget gates above.

$$\mathbf{ç}_t = \mathbf{f}_t \mathbf{ç}_{t-1} + \mathbf{i}_t \tanh\left(\mathbf{W}_c^x \mathbf{x}_t + \mathbf{W}_c^h \mathbf{h}_{t-1} + \mathbf{b}_c\right) \tag{4.9}$$

Similarly, an output gating function is used to control the impact of the memory cell contents on the current hidden vector representation, and hence the prediction at the current time step.

$$\mathbf{o}_t = \sigma\left(\mathbf{W}_o^x \mathbf{x}_t + \mathbf{W}_o^h \mathbf{h}_{t-1} + \mathbf{b}_o\right) \tag{4.10}$$

$$\mathbf{h}_t = \mathbf{o}_t \tanh\left(\mathbf{ç}_t\right) \tag{4.11}$$

Figure 4.3 provides an indication of the information flow described by the set of equations presented above. The extensive use of gating is a better solution to the problem of vanishing and exploding gradients than TBPTT alone, hence the improved context over long ranges. Variations on the LSTM theme such as Gated Recurrent Units (GRUs) [40] and the introduction of peephole connections [41] have been shown to provide gains on a task specific basis.

As was the case for RNNs, a natural extension to LSTMs is to arrange them in a bidirectional fashion. If a single LSTM unit is represented by the function $\text{LSTM}(\cdot)$, then the information propagation in a bidirectional LSTM network can be described by equations 4.12 to 4.15.

$$\left[\begin{array}{cc} \overrightarrow{\mathbf{h}}_t & \overrightarrow{c}_t \end{array}\right]^\top = \text{LSTM}\left(\overrightarrow{\mathbf{h}}_{t-1},\, \mathbf{x}_t,\, \overrightarrow{c}_{t-1}\right) \tag{4.12}$$

$$\left[\begin{array}{cc} \overleftarrow{\mathbf{h}}_t & \overleftarrow{c}_t \end{array}\right]^\top = \text{LSTM}\left(\overleftarrow{\mathbf{h}}_{t+1},\, \mathbf{x}_t,\, \overleftarrow{c}_{t+1}\right) \tag{4.13}$$

$$\mathbf{h}_t = \left[\begin{array}{cc} \overrightarrow{\mathbf{h}}_t & \overleftarrow{\mathbf{h}}_t \end{array}\right]^\top \tag{4.14}$$

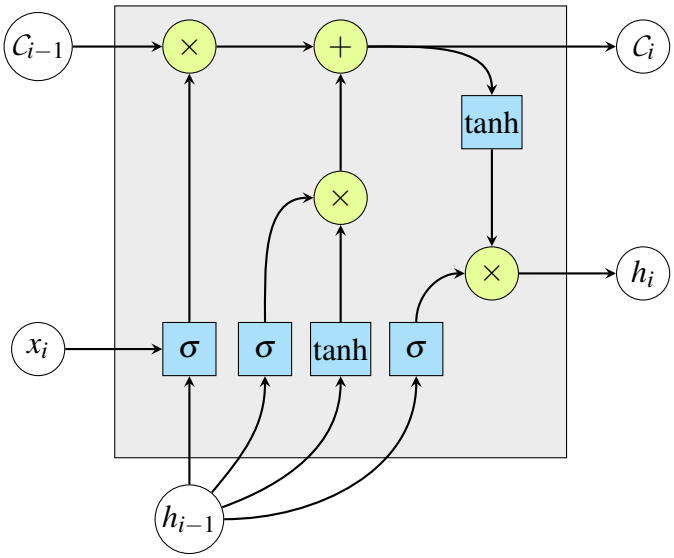


Fig. 4.3 A diagram illustrating the flow of information in a single LSTM unit

$$y_t = f_y(\mathbf{h}_t) \tag{4.15}$$

4.1.4 Gated Recurrent Unit

The Gated Recurrent Unit (GRU) is an adaptation of the LSTM unit which uses a similar gating strategy to capture long term time dependencies along a sequence [42]. Rather than the three gating functions used by the LSTM, the GRU only consists of two gates: the output and forget gates. Without an input gate, the GRU exposes the full content of a new input to the memory cell. An advantage of this is the reduced number of parameters, which translates into a simplified optimisation process. A standard configuration for the GRU is given in equation 4.16 through to equation 4.19. The two gating functions are defined as

$$\mathbf{i}_t = \sigma\left(\mathbf{W}_i^x \mathbf{x}_t + \mathbf{W}_i^h \mathbf{h}_{t-1} + \mathbf{b}_i\right) \tag{4.16}$$

$$\mathbf{f}_t = \sigma\left(\mathbf{W}_f^x \mathbf{x}_t + \mathbf{W}_f^h \mathbf{h}_{t-1} + \mathbf{b}_f\right) \tag{4.17}$$

while the candidate activation is computed as

$$\tilde{\mathbf{h}}_t = \tanh\left(\mathbf{W}_h^x \mathbf{x}_t + \mathbf{W}_h^r \left(\mathbf{f}_t \odot \mathbf{h}_{t-1} + \mathbf{b}_h\right)\right) \tag{4.18}$$

where $\odot$ is an element-wise multiplication operation. The activation, 4.19, of a GRU layer at time index t is given by a linear interpolation between the candidate activation and the activation at the $t-1$.

$$\mathbf{h}_t = (\mathbf{1} - \mathbf{i}_t) \odot \mathbf{h}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{h}}_t \tag{4.19}$$

The set of operations above which describe a GRU can be simplified in a similar manner as was the case for LSTMs. A bidirectional GRU (BiGRU) can therefore be described as

$$\overrightarrow{\mathbf{h}}_t = \text{GRU}\left(\overrightarrow{\mathbf{h}}_{t-1},\ \mathbf{x}_t\right) \tag{4.20}$$

$$\overleftarrow{\mathbf{h}}_t = \text{GRU}\left(\overleftarrow{\mathbf{h}}_{t+1},\ \mathbf{x}_t\right) \tag{4.21}$$

$$\mathbf{h}_t = \left[\begin{array}{cc} \overrightarrow{\mathbf{h}}_t & \overleftarrow{\mathbf{h}}_t \end{array}\right]^\top \tag{4.22}$$

where GRU$(\cdot)$ describes the set of equations which make up the GRU unit. The predictions are obtained from a non-linear mapping of the hidden vector.

$$y_t = f_y(\mathbf{h}_t) \tag{4.23}$$

The similarities between the LSTM and the GRU make it difficult to predict which structure is expected to perform better in the general case. The design decision between using a GRU or LSTM is almost entirely dependent on empirical results for each particular application [40].

4.2 Attention

A significant proportion of recent advancements in sequence modelling has been due to the introduction of attention as a tool for generating a context vector which has access to the entire input sequence. This avoids the need to explain the entire history of the sequence in a single hidden state since this is often problematic given a finite hidden state size. As a result, attention-based models are able to learn alignments between different modalities rather than operating in a linear fashion and propagating information sequentially as is the case in traditional sequence to sequence models [43]. A general form of attention views the context vector for a T length sequence, $\mathbf{v}$, as a simple averaging operation weighted by the parameter $\boldsymbol{\alpha}$. This sequence is often referred to as the *value* when discussing attention mechanisms.

$$\mathbf{c} = \sum_{t=1}^{T} \alpha_t \mathbf{v}_t \tag{4.24}$$

The parameter vector $\boldsymbol{\alpha}$, commonly referred to as the attention weights, is normalised over the sequence using a softmax function.

$$\alpha_t = \frac{\exp(e_t)}{\sum_{t=1}^{T} \exp(e_t)} \tag{4.25}$$

The attention weights before normalisation, e_t, are derived by a similarity or scoring function where the interactions between a key, $\mathbf{k}_t$ and a query, $\mathbf{q}_t$, determine the weighting of each element across the sequence. A general functional form for this similarity function is presented below.

$$e_t = f_{sim}\left(\mathbf{k}_t,\ \mathbf{q}_t\right) \tag{4.26}$$

4.2.1 Additive Attention

An early form of similarity function, defined in equation 4.27, led to the popularisation of additive attention, which was invented as a mechanism for generating sequence alignments in machine translation [44]. A unique aspect of this similarity function, is that the key and query have independent learnable matrices $\mathbf{A_k}$ and $\mathbf{A_q}$.

$$e_t = \mathbf{w}^T \tanh(\mathbf{A_k k} + \mathbf{A_q q}) \tag{4.27}$$

A slight variation on the original additive attention is presented in [43] where the key and query are concatenated before undergoing a single linear transformation. This change, presented in equation 4.28, results in a simpler computation path.

$$e_t = \mathbf{w}^T \tanh(\mathbf{A}\left[\mathbf{k},\ \mathbf{q}\right]^T) \tag{4.28}$$

4.2.2 Multiplicative Attention

Another form of attention simplifies the similarity function to a matrix multiplication between the key, a trainable weight matrix $\mathbf{A}$, and the query as shown in equation 4.29. This has practical benefits over additive attentions in terms of speed and memory efficiency. This similarity function was proposed in [43] under the name of general attention, however multiplicative attention is a more intuitive description.

$$e_t = \mathbf{k}_t{}^T \mathbf{A}\ \mathbf{q}_t \tag{4.29}$$

4.2.3 Dot Product Attention

The scoring function given in equation 4.29 can be simplified to the dot-product of the key with the query by setting **A** to be an identity matrix. This form of scoring function is often scaled by the square root of the number of features in a scheme which is called scaled dot product attention [33]. The scaling operation aims to ensure that the input to the softmax normalisation is small enough such that the function is operating in a region where the gradient is large enough for reasonable weight updates during training. The dot product and scaled dot product similarity functions are

$$e_t = \mathbf{k}_t{}^T \mathbf{q}_t \tag{4.30}$$

$$e_t = \frac{1}{\sqrt{dim(\mathbf{k}_t)}} \mathbf{k}_t{}^T \mathbf{q}_t \tag{4.31}$$

where $dim(\mathbf{k}_t)$ is the number of features in the key.

4.2.4 Self-Attention

Self-attention is a particular form of attention where the key, query, and value all take on the same value. Self-attention has the additional advantage of being easily parallelized and the computational complexity per layer is less than that of a recurrent network [33]. The self-attention model using the form of similarity described in equation 4.29 yields equation 4.32 where the key, value, and query are represented by $\mathbf{v}_t$.

$$e_t = \mathbf{v}_t{}^T \mathbf{A}\ \mathbf{v}_t \tag{4.32}$$

If the **A** matrix is set to the identity, the similarity function reduces to the L_2-norm operation. Work done in [45] indicates that the Transformer model, which exploits scaled dot-product self-attention, has strong semantic feature extraction ability. This is indicative that although the similarity matrix is simple, it still has predictive power.

Chapter 5

Deep Learning for Confidence Scores

In this chapter the deep learning tools described in chapter 4 are discussed in terms of their applications to confidence scores. Recurrent networks for estimating confidence scores on one-best lists are considered before introducing LatticeRNN as a generalisation for lattices and confusion networks. Thereafter, a scheme for incorporating sub-word level information into LatticeRNN is presented along with a description of the methods for obtaining the sub-word level information from the decoded output. The training criterion for the model as well as the mechanisms for labelling the lattices, confusion networks, and one-best sequences with the target confidence scores are presented.

5.1 BiLSTM for Confidence Scores on One-Best Sequences

As implied by the name, one-best sequences consist of a series of arcs connected consecutively to form a sequence. For each arc in the sequence, a confidence score prediction is desired. This is a fairly conventional setup for a sequence-to-sequence problem. It follows that a BiLSTM network, presented in graphical form in figure 5.1, is constructed such that information is propagated forwards and backwards through the sequence in an end-to-end deep learning framework. For a given arc, e_i, the hidden state representation in the forward and backwards directions are given by equations 5.1 and 5.2 respectively where $\mathbf{x}_i$ is the feature vector for e_i.

$$\overrightarrow{\mathbf{h}}_i = \text{LSTM}\left(\overrightarrow{\mathbf{h}}_{i-1},\ \mathbf{x}_i\right) \tag{5.1}$$

$$\overleftarrow{\mathbf{h}}_i = \text{LSTM}\left(\overleftarrow{\mathbf{h}}_{i+1},\ \mathbf{x}_i\right) \tag{5.2}$$

The confidence score prediction, c_i, is a function of both the forward and backward states as demonstrated in equation 5.3.

$$c_i = f_c\left(\overrightarrow{\mathbf{h}}_i, \overleftarrow{\mathbf{h}}_i\right) \tag{5.3}$$

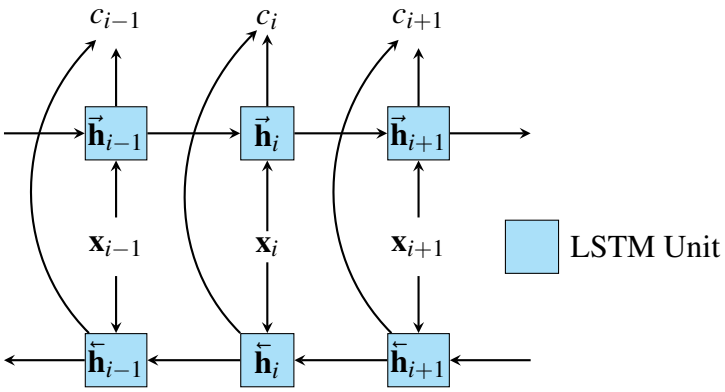


Fig. 5.1 BiLSTM for predicting confidence scores over a one-best sequence.

Embeddings

In order to insert lexical information, such as words or sub-words, into a mathematical model, an embedding is required. Word embeddings are a low dimensional vector made up of continuous elements for which there exists a one-to-one mapping between each word in the vocabulary to a vector in the embedding space. A key property of a word embedding as opposed to a simple one-hot encoding is that the embedding size is not directly dependent on the vocabulary size, which is often too large to make a one-hot representation useful in practice. Word embeddings generated using neural networks explicitly encode linguistic patterns observed in the training corpus. As a result, it is often interesting to investigate the clustering and relative distance metrics between words in the embedding space [46]. For the word embeddings used in this work, the 50-dimensional word embeddings trained in [26] are used. These were generated using the `fastText` model [47]. `fastText` has a number of advantages over `word2vec` because it views each word as a collection of character n-grams. This means that `fastText` is able to generate better word embeddings for rare words, since the respective n-grams are likely to be more common, and embeddings for out of vocabulary words are possible.

5.2 LatticeRNN for Confidence Scores

Simply put, LatticeRNN is a generalisation of RNNs from the simple case of one-best sequences to lattices and confusion networks. By taking advantage of the fact that graph-like topologies share common substructures between hypotheses, LatticeRNN is able to efficiently encode the full graph in an architecture which is suitable for applying end-to-end deep learning to a variety of problems. The flexibility provided by being able to make predictions directly on the full lattice, rather than introducing destructive decisions early in the prediction process, can lead to performance gains [48]. An overview of the LatticeRNN model for confidence scores, is provided in figure 5.2.

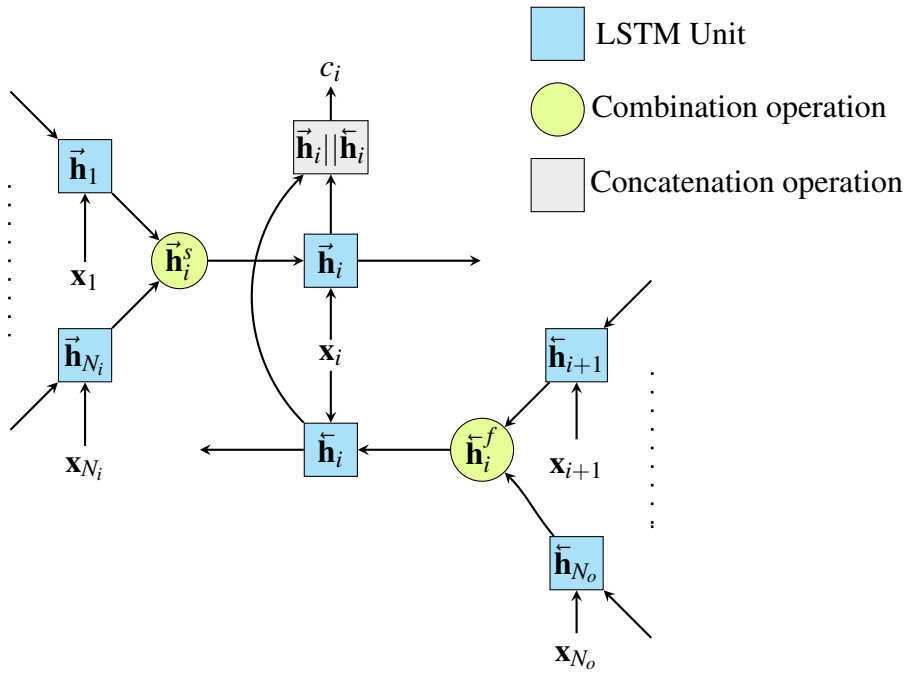


Fig. 5.2 Bi-Directional LatticeRNN model for confidence score prediction.

5.2.1 Architecture

The LatticeRNN model is able to operate on any DAG by traversing the network in topological order [48]. Arc posterior information is propagated in a forward-backward style whilst making use of a bi-directional LSTM to maintain long term dependencies over the length of the set of candidate sequences. Each arc, e_i, has a feature vector, $\mathbf{x}_i$, corresponding to the word which that arc represents. Additionally, each arc has a forward and backward

hidden state vector given by $\vec{\mathbf{h}}_i$ and $\overleftarrow{\mathbf{h}}_i$ respectively. The number of features making up $\mathbf{x}_i$ can be arbitrary, therefore it is common to have multiple features associated with each arc as described in section 2.4.1.

The most significant deviation from the forward-backward algorithm is in how multiple hidden state vectors are combined when multiple arcs terminate at the same node. A number of arc merging strategies have been investigated with attention yielding the best performance [1]. For a set of incoming arcs, $\{e_1, e_2, \ldots, e_{N_i}\}$, to arc, e_i, the merged hidden state vector in the forward direction is formulated by equation 5.4. This merged state vector, $\vec{h}_i^f$, is then used as input to the LSTM unit which generates the hidden state representation for the arc in the forward direction, $\vec{\mathbf{h}}_i$, as given in equation 5.5.

$$\overrightarrow{\mathbf{h}}_i^s = \sum_{j=1}^{N_i} \alpha_j \, \overrightarrow{\mathbf{h}}_j \tag{5.4}$$

$$\overrightarrow{\mathbf{h}}_i = \text{LSTM}\left(\overrightarrow{\mathbf{h}}_i^s, \, \mathbf{x}_i\right) \tag{5.5}$$

The backward pass applies a similar arc merging scheme where given the set of all arcs diverging from a node, $\{e_{i+1}, e_{i+2}, \ldots, e_{N_o}\}$, the backwards hidden state vector formulated by the combination of these arcs is given by equation 5.6 and the backwards recurrence is defined by equation 5.7.

$$\overleftarrow{\mathbf{h}}_i^f = \sum_{j=i+1}^{N_o} \alpha_j \overleftarrow{\mathbf{h}}_j \tag{5.6}$$

$$\overleftarrow{\mathbf{h}}_i = \text{LSTM}\left(\overleftarrow{\mathbf{h}}_i^f, \, \mathbf{x}_i\right) \tag{5.7}$$

Unlike CRFs each node and edge is encoded into its own hidden state, $\mathbf{h}_i$, through the concatenation of $\vec{\mathbf{h}}_i$ and $\overleftarrow{\mathbf{h}}_i$.

$$\mathbf{h}_i = \begin{bmatrix} \overrightarrow{\mathbf{h}}_i & \overleftarrow{\mathbf{h}}_i \end{bmatrix}^\top \tag{5.8}$$

The predicted confidence score, c_i, for a particular arc e_i is simply a function of the concatenated hidden state vector.

$$c_i = f_c(\mathbf{h}_i) \tag{5.9}$$

5.2.2 Attention for Arc Merging

The form of attention used to formulate the weighted distribution, $\boldsymbol{\alpha}$, is a modified version of the additive attention mechanism presented in section 4.2.1. The key and query are concatenated and then linearly transformed by a learnable weight matrix, $\mathbf{A}$. The resulting vector is fed into an element-wise ReLU activation. The dot product between a separate learnable column vector, $\mathbf{w}$, and the ReLU output generates a scalar value. The final step in the arc merging similarity function as implemented in [1] applies a $\tanh(\cdot)$ activation to this scalar output.

The key, $\mathbf{k}_j$, is selected to be a vector made up of the incoming arc posterior $p(e_j|\mathbf{O},\lambda)$ concatenated with the mean, μ_j, and variance, σ_j, of the arc posteriors for the set of arcs over which the attention mechanism is operating. The query, $\mathbf{q}_j$, is simply defined as the hidden state vector, $\mathbf{h}_j$. As a result, the arc merging similarity function can be defined by

$$e_j = \tanh\left(\mathbf{w}^T \text{ReLU}(\mathbf{A}\left[\mathbf{k}_j,\ \mathbf{q}_j\right]^T)\right) \tag{5.10}$$

where $\mathbf{k}_j = \left[p(e_i|\mathbf{O},\lambda),\ \mu_j,\ \sigma_j\right]$ and $\mathbf{q}_j = \mathbf{h}_j$. As usual, a softmax function is used to normalise the attention weights into a valid probability distribution over the hidden vector.

5.3 Sub-word Information for LatticeRNN

The LatticeRNN model described above does not take into account any sub-word level features when producing predictions. This section describes an extension for incorporating grapheme features into the LatticeRNN framework.

5.3.1 Grapheme Features

Due to the dynamic word length across the language, the grapheme information per-word can be represented as a variable length sequence of grapheme feature vectors. It is important that the grapheme information fits into the the existing LatticeRNN framework. One way to achieve this is to append the new grapheme features to the existing feature vector containing the word information, $\mathbf{x}_i$.

For each word on an arc e_i, a grapheme sequence feature vector, $\mathbf{g}_i$ is defined as a variable length sequence $\mathbf{g}_i = \left\{\mathbf{g}_i^{(1)},\ \mathbf{g}_i^{(2)},\ \ldots,\ \mathbf{g}_i^{(J_i)}\right\}$ where J_i is the number of graphemes in the word associated with the arc e_i. In this work, two grapheme level features are considered. These are the grapheme duration and the grapheme itself. The feature vector for the j^{th} grapheme on arc e_i is defined as

$$\mathbf{g}_i^{(j)} = \begin{bmatrix} \mathbf{s}_i{}^{(j)} \\ d_i^{(j)} \end{bmatrix} \tag{5.11}$$

where $\mathbf{s}_i^{(j)}$, is an embedding of the grapheme and $d_i^{(j)}$ is the grapheme duration.

Grapheme Embeddings

Georgian is a non-Latin language which contains 33 graphemes, of which 5 are vowels and 28 are consonants. Six additional characters are included when generating embeddings of the Georgian graphemes: `<s>`, `</s>`, `sp`, `sil`, `G00`, and `G01`. The first two symbols are the start and end of sentence tags. It was decided to let a single embedding be allocated to both of these tokens. This design decision means that the grapheme embedding does not distinguish between these two contexts. The `sp` and `sil` characters denote short pause and silence while `G00` and `G01` indicate hesitations in speech. The small size of the grapheme vocabulary circumvents many of the common challenges that large vocabulary embeddings face, such as the limited number of training examples for rare words. Another advantage is that since the set of graphemes in a language is strictly defined, there is no need to handle out of vocabulary tokens. For these reasons, the advantages of `fastText` are not pertinent when generating grapheme embeddings. As a result, `Word2Vec` is used to generate a 4-dimensional Continuous Bag of Words (CBOW) grapheme embedding [46].

Figure 5.3 is a t-SNE projection of the grapheme embeddings from 4-dimensions to 2-dimensional space [49]. A perplexity of 5, learning rate of 200, and 20 000 iterations was used to generate this visualisation. An interesting result from this is that the cluster of graphemes circled in red contains four of the five Georgian vowels. This is an indication that the embedding has learnt some useful structure.

In order to include this information in the LatticeRNN framework, a fixed form representation of the grapheme sequence features is required. To this end, an attention-based grapheme feature merging technique is explored.

5.3.2 Grapheme Feature Merging

Various attention mechanisms for merging grapheme features into fixed form representations are investigated. The intuition behind the decision to use attention for this task is that particular graphemes in a word may be more relevant than others for determining the confidence associated with a particular arc. This relationship could be best learnt from an adaptable aggregation process such as attention. Furthermore, this decision removes the need

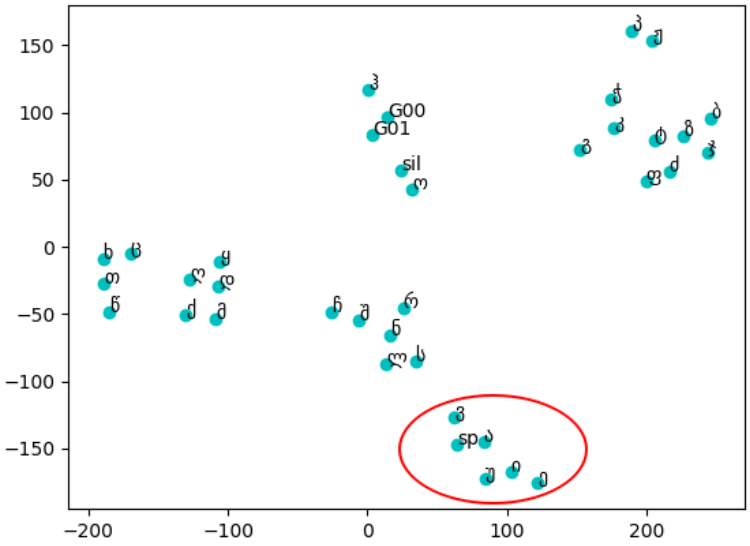


Fig. 5.3 t-SNE visualisation of the grapheme embeddings.

to obtain expert knowledge of the Georgian language which would be required to apply a rule based approach. A number of attention-based merging procedures are introduced below. These can be separated into two classes: a 'flat' attention mechanism operating directly over the grapheme sequence and an encoder-based model where the attention operates over the hidden states of a recurrent encoding of the grapheme features.

Flat Grapheme Attention

Inspired by the transformer, this approach does away with sequence aligned recurrency and convolutions. The aim of the 'flat' attention mechanism is to generate rich representations of the grapheme feature sequence by attending over the grapheme sequence directly. Figure 5.4 provides an overview of the structure of this model where the attn$(\cdot)$ block indicates the attention operation and $\tilde{\mathbf{g}}_i$ is the merged representation of the grapheme information.

Four attention schemes were applied, the first of these being multiplicative self-attention which has a similarity function which takes on the form presented in equation 5.12. This can be viewed as a generalised norm operation of the grapheme feature sequence.

$$e_i^{(j)} = \mathbf{g}_i^{(j)^T} \mathbf{A}\, \mathbf{g}_i^{(j)} \tag{5.12}$$

The second form of attention implemented, scaled dot product self-attention, is a special case of multiplicative attention where A is an identity matrix, hence the similarity function

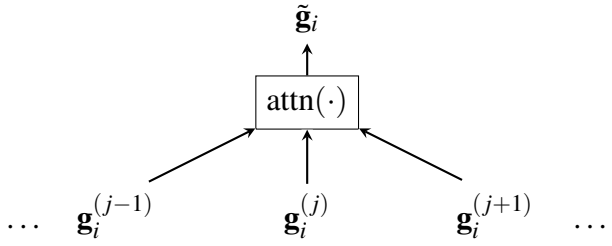


Fig. 5.4 'Flat' attention over the grapheme feature sequence.

is the L_2-Norm of the grapheme feature vector. This simply means that the greater the magnitude of the grapheme feature vector, the greater the contribution of that grapheme to the fixed form representation.

$$e_i^{(j)} = \frac{1}{\sqrt{dim(\mathbf{g}_i^{(j)})}} \mathbf{g}_i^{(j)^T} \mathbf{g}_i^{(j)} \tag{5.13}$$

A variant of additive attention as described by [43] is used in a self-attention configuration. The following expression is presented in equation 5.14. In addition to the linear transformation, the grapheme feature undergoes a non-linear mapping followed by a second linear transformation.

$$e_t = \mathbf{w}^T \tanh(\mathbf{A} \left[\mathbf{g}_i^{(j)}, \mathbf{g}_i^{(j)}\right]^T) \tag{5.14}$$

In contrast to the three similarity functions described above, the final form of attention uses a key which is the concatenation of the grapheme feature vector and the word duration, $t_i^{(d)}$. The resulting key is defined by

$$\mathbf{k}_i^{(j)} = \begin{bmatrix} \mathbf{g}_i^{(j)} \\ t_i^{(d)} \end{bmatrix} \tag{5.15}$$

The rational for this choice is that the relationship between the duration of each grapheme and the duration of the full word aught to provide insight into the significance of that particular grapheme for confidence score estimation. Introducing this key into the Luong additive attention similarity function yields the following result.

$$e_t = \mathbf{w}^T \tanh(\mathbf{A} \left[\mathbf{k}_i^{(j)}, \mathbf{g}_i^{(j)}\right]^T) \tag{5.16}$$

In all four of these models, the fixed length representation takes on the length of a single grapheme feature.

Encoded Grapheme Attention

An alternative to applying attention over the grapheme features directly, is to encode the features using a recurrent structure and apply the attention over the hidden vectors. This approach results in a richer representation of the grapheme features being developed before merging into a reduced form. To this end, a bi-directional recurrent structure, as presented in figure 5.5, is constructed where the concatenated hidden encoder state for grapheme j on arc i is represented by $\mathbf{z}_i^{(j)}$. Three forms of bi-directional recurrent unit are explored for this purpose. These are the standard BiRNN, BiGRU, and BiLSTM units. The aim of the encoder is to encapsulate contextual information in the grapheme sequence. The size of the fixed form representation is determined by the dimensions of the hidden vector in the encoder, $\mathbf{z}_i^{(j)}$.

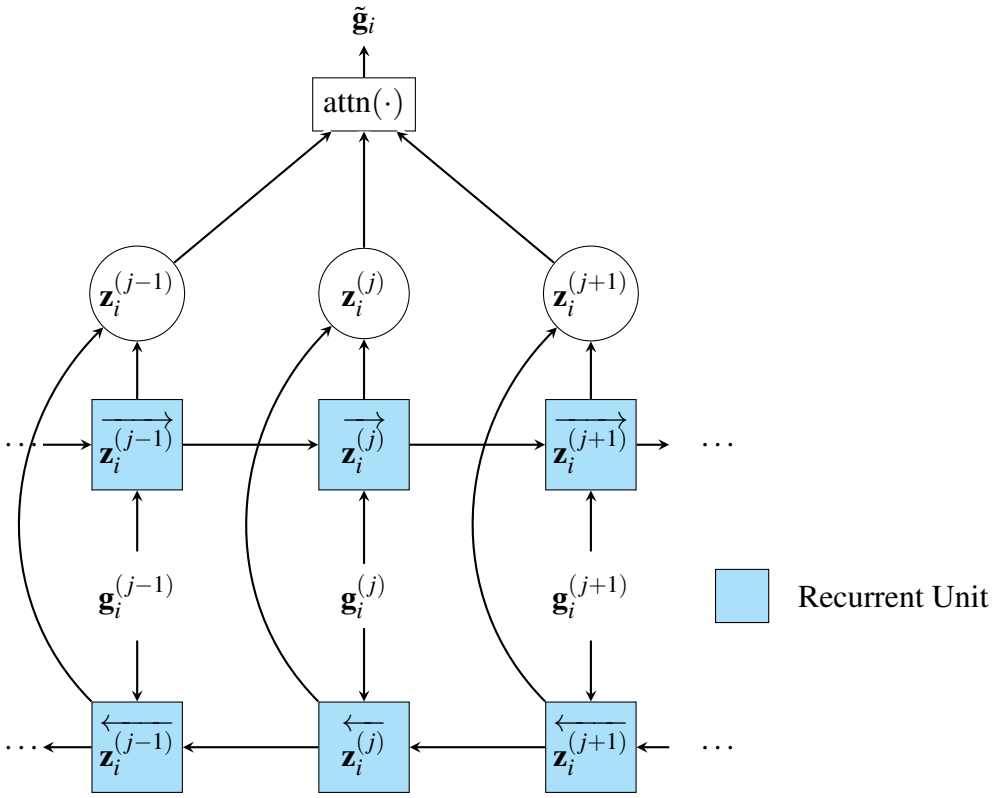


Fig. 5.5 Bi-directional recurrent encoding of the grapheme sequence.

The grapheme information is incorporated into the LatticeRNN framework by concatenating the fixed form representation with the word information as presented in figure 2.5. Figure 5.6 illustrates the extension for the case where the grapheme information is combined

with the word information alongside the word posterior probability, p_i. Although the start time of each arc is available, it is not included in the feature vector as it does not provide predictive power for determining confidence scores

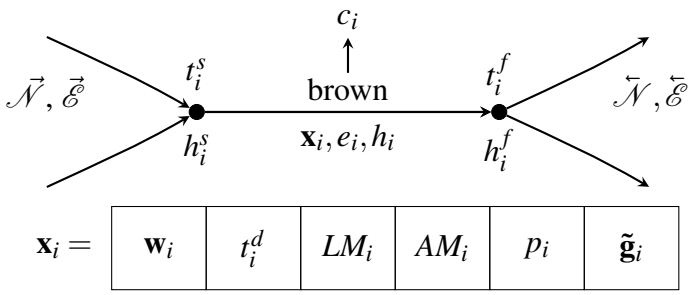


Fig. 5.6 The edge for the word 'brown' with the arc posterior probability and merged grapheme information included.

5.3.3 Extracting Grapheme Features

As introduced in section 2.4, lattices, confusion networks, and one-best sequences, do not inherently contain the same set of features. The arc clustering process used to generate confusion networks from lattices, as described in chapter 2, is an example of a process which results in information being sacrificed when moving to a more constrained structure. With this in mind, there are a number of ways in which the grapheme features can be obtained from a decoded output. Figure 5.7 demonstrates this for each of the structures described in section 2.4.

For lattices generated from a graphemic ASR system, this is a simple procedure involving a determinisation operation on the lattice. For confusion networks, an approximate solution is achievable by pairing each confusion network with the grapheme-marked lattice which models the same utterance. An arc matching algorithm, described in section 5.3.4, is used to find the best matching arc in the lattice for each arc in the confusion network. This provides the confusion network with an approximate grapheme-marking. For one-best sequences, three possible methods were considered based on how the one-best sequence in question was generated. For a one-best sequence extracted from a lattice via the Viterbi algorithm, a forced alignment between that one-best sequence and the graphemes in the observation sequence provides the one-best sequence with the desired grapheme features. The same operation can be applied to one-best sequences generated from confusion networks, however these one-best sequences do not have access to the LM and AM scores which were lost during the lattice to confusion network stage. Finally, an approximate grapheme-marking can be obtained by applying the arc matching algorithm to the confusion network generated one-best sequences.

The grapheme-marked one-best sequences used in the experiments presented in chapter 7 were generated through a forced alignment of the MBR generated one-best sequences.

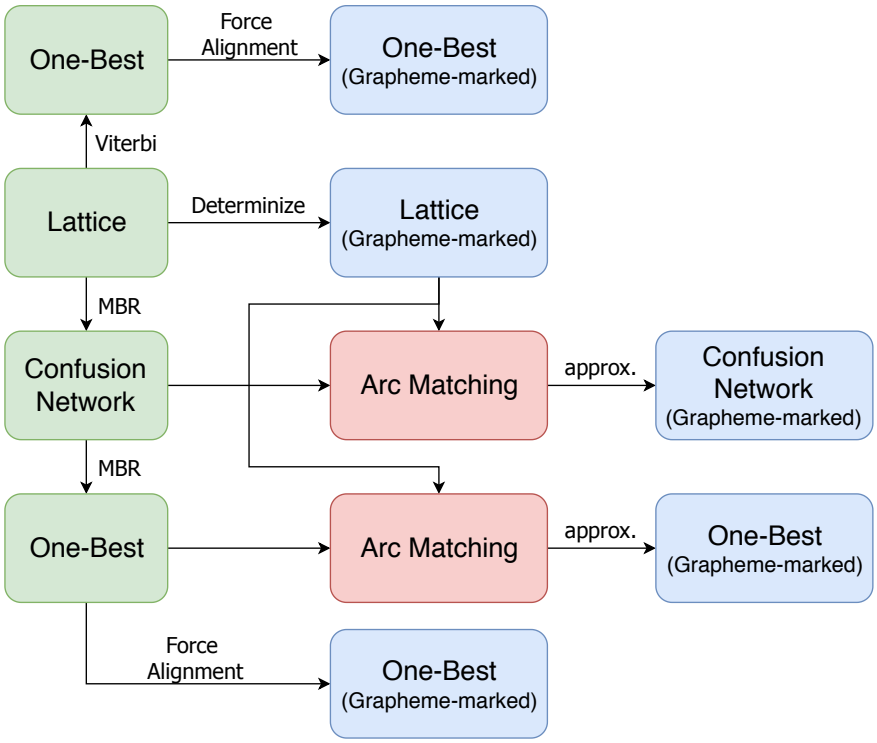


Fig. 5.7 The different mechanisms to obtain grapheme features from the decoding process.

5.3.4 Arc Matching For Feature Sharing

An algorithm which determines the cost between a particular arc in a graphical structure and all the arcs in a separate graph-like structure is presented. This loss is defined using the L_2-Norm as presented in equation 5.17 for all arc pairs which have the same word and have a difference in start or end time of less than 1.5 seconds. Arcs which do not meet these initial thresholds are rejected from the outset. All candidate arcs are ranked according to L_2-Norm cost and ties are ranked using decreasing order of arc posterior probability. All start and end time pairs which have a time difference error of five or fewer frames are considered to be an exact match. The audio recordings have time markings which are accurate to the system frame period of 0.01 seconds.

$$L_2\text{-Norm} = \sqrt{\left(t_i^s - t_j^s\right)^2 + \left(t_i^f - t_j^f\right)^2 + \left(t_i^d - t_j^d\right)^2} \tag{5.17}$$

In addition to extracting approximate grapheme features as presented above, the LM, AM, and word duration of the top ranked lattice arc can be inserted into the confusion network or one-best sequence. In some cases, an arc may not have a reasonable match in the corresponding lattice. There are a few options for dealing with this scenario, such as assigning an equal duration to each grapheme in the word. In this work, audio segments containing rejected arcs are omitted from the dataset. These cases are clearly marked in the experimental results chapter when they occur. Figure 5.8 provides an overview of the arc matching process where the grey objects are system inputs and the blue objects are the resulting grapheme-marked structures.

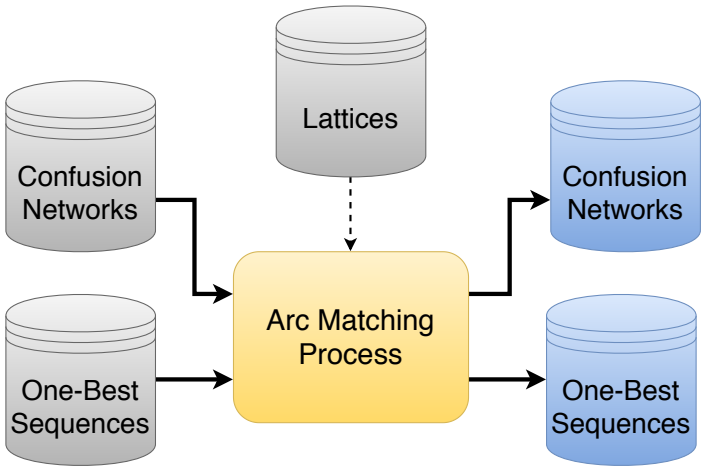


Fig. 5.8 An illustration of the information flow in the arc matching process.

5.4 Training Criterion

At the core of any deep learning approach, is the iterative process of optimising a loss as defined by a training criterion. In order to construct such a training criterion, it is important to ensure that all arcs are labelled with an appropriate confidence score target. To this end, an idealised word-based confidence score is defined as a binary classification problem where c_t is the reference confidence score and $\bar{c}_t$ is the target confidence score. The predicted word output from the ASR system is used to generate these confidence score tags as demonstrated by equation 5.18, where $\bar{y}_t$ is the reference word and y_t is the predicted word for arc e_i in the sequence.

$$\bar{c}_i = \begin{cases} 1 & \text{if} \quad \bar{y}_i = y_i \\ 0 & \text{otherwise} \end{cases} \tag{5.18}$$

For a binary classification task, a binary cross-entropy loss over all arcs can be minimised where $\bar{y}_i$ is the reference label which denotes the ground truth and y_i is the predicted classification.

$$\mathscr{L} = -\sum_{i=1}^{N} \bar{y}_i \log(y_i) + (1-\bar{y}_i)\log(1-y_i) \tag{5.19}$$

Alternatively, one can choose to optimise with respect to the arcs in the one-best sequence only. This is particularly relevant for applications where confidence scores on the one-best sequence are prioritised. If the arc indices making up the one-best sequence are defined as the set $\mathscr{O}$, then the expression for binary cross entropy over the one-best sequence is given by

$$\mathscr{L}_{\mathscr{O}} = -\sum_{i\in\mathscr{O}} \bar{y}_i \log(y_i) + (1-\bar{y}_i)\log(1-y_i) \tag{5.20}$$

5.5 Arc Tagging

A crucial component of the optimisation process is the generation of ground truth targets. The subsections to follow describe how to determine whether the seemingly simple equality, $\bar{y}_i = y_i$, is met. This is a critical aspect of the tagging process since not only should the respective words be an exact match, but the alignments between the predicted and reference words must concur.

5.5.1 Tagging One-Best Sequences

Owing to the strict sequential nature of a one-best sequence, each arc can be tagged directly using the Levenshtein distance defined in section 2.5. The mathematical formulation of the Levenshtein distance treats each of the three types of error distinctly. Insertion and substitution error types are represented in a one-best hypothesis in the form of an extra arc for insertion errors, and an arc containing an incorrect word for substitution errors. As discussed in chapter 3, the absence of an arc is unable to be represented in a one-best hypothesis thus a confidence score cannot be assigned for deletion errors [31].

5.5.2 Tagging Confusion Networks

In a similar manner to the one-best sequences, the Levenshtein distance metric can be applied directly to each of the alignments in a confusion network. One advantage of the confusion network topology is that since all arcs must pass through each node, an exact distance

measure can still be applied whilst not sacrificing the extra information from the confusion arcs.

5.5.3 Approximate Levenshtein for Tagging Lattices

The flexible structure of lattices requires that the start and end times associated with each arc are considered when determining whether to tag an arc as incorrect or correct. If the start and end times for the word level aligned reference are given by r_s and r_e and the start and end times for a particular candidate word on an arc are given by t_s and t_e, the degree of overlap is defined by equation 5.21. By setting a reasonable threshold for the minimum overlap ratio required for an arc to be considered correctly aligned, η_{thresh}, an approximation of the Levenshtein arc tagging algorithm can be used by checking for the condition $\eta > \eta_{thresh}$.

$$\eta = max\left\{0,\ \frac{\|min\{r_e,t_e\}\| - \|max\{r_s,t_s\}\|}{\|max\{r_e,t_e\}\| - \|min\{r_s,t_s\}\|}\right\} \tag{5.21}$$

Although the Levenshtein approach can be used to tag the one-best path in a lattice, this Levenshtein approximation is particularly useful when all arcs in the lattice need to be tagged [1].

Chapter 6

Implementation

To demonstrate the proposed model for improving confidence scores, a practical implementation is required. This chapter describes the underlying data which is used to train the confidence score prediction models presented in the experiments to follow. This includes processing the HTK generated files into an efficient structure for training. The vast majority of the implementation is carried out in the Python software language. Extensively used libraries include PyTorch [50] and NumPy [51]. These were selected for easy compatibility with the existing LatticeRNN code base and the flexibility and readability that PyTorch offers compared to other machine learning libraries such as Tensorflow. The dynamic computational graphs employed by PyTorch are useful in a research settings as it allows for quick and easy debugging. This chapter presents the data processing pipeline and details on the training procedure.

6.1 Data Processing

The lattices, confusion networks, and one-best sequences are not stored to disk in a convenient form to be directly processed by traditional deep learning framework. For this reason, a data processing pipeline is constructed. The data pipeline used leverages the existing code implemented in [26] with a number of modifications and extensions. Figure 6.1 gives a high level overview of the data processing pipeline which converts the raw lattice, confusion networks, and one-best sequences into a NumPy file format which is suitable for training. Furthermore, target confidence arc tags are produced for the full structure as well as the one-best sequence.

The objective of the data processing pipeline is to convert the native compressed HTK-style lattice (`.lat.gz`) and confusion network (`.scf.gz`) files into a single consistent format which can be easily and efficiently loaded by the `PyTorch` model. For each object processed

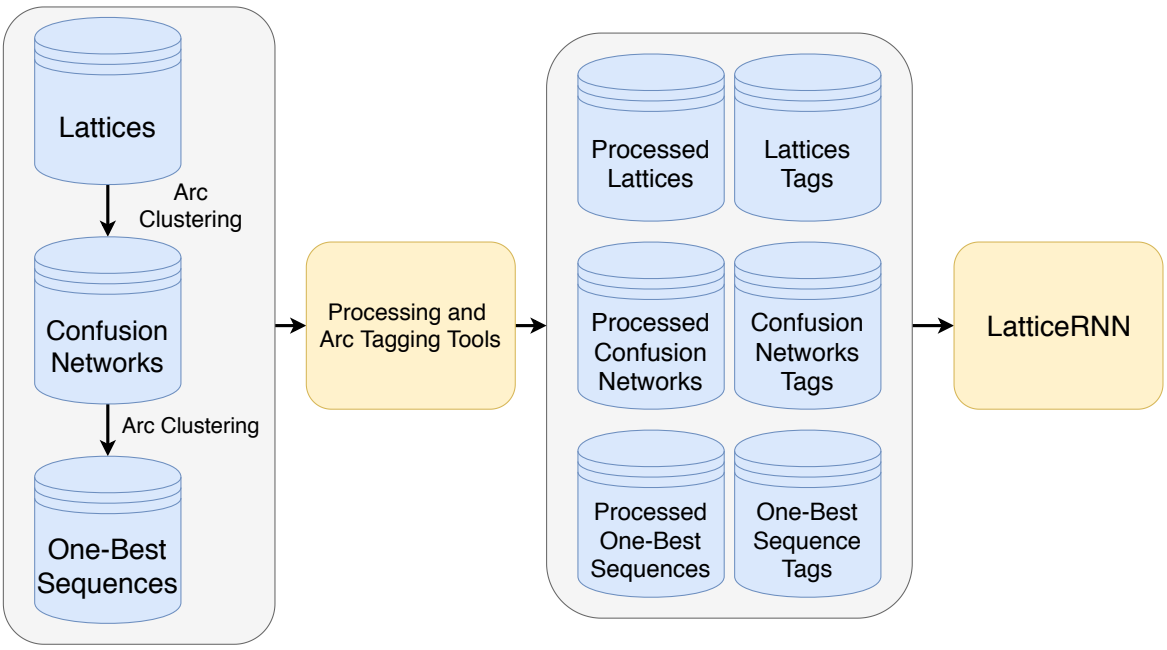


Fig. 6.1 The data processing pipeline.

by the pipeline, two files are produced: a processed `.npz` object representation of the lattice or confusion network and a corresponding `.npz` object containing the target confidence scores. The processed lattice or confusion network file contains all the original attributes described in [26] as well as the following additional attributes:

- `grapheme_data`: A tensor containing the grapheme features for each grapheme in the corresponding lattice or confusion network object. In order to store and process the tensor efficiently, the dimension corresponding to the grapheme sequence length is padded to match the length of the longest grapheme sequence in the lattice or confusion network.
- `start_times`: A list of the start time of each arc (in seconds) ordered consistently with the `edge_data` attribute.

The target files have the following attributes:

- `target`: A list of the ground truth target confidence scores for each arc in the lattice or confusion network.
- `indices`: A list of the arc indices which make up the one-best sequence. The indices are given in the expected sequential order.
- `ref`: A list of ground truth target confidence scores for each arc in the one-best sequence.

As part of the processing procedure, data whitening is employed to shift the mean of each feature to zero and scale the standard deviation to one. Due to the large amount of data making up the corpus, an online algorithm is used to compute the mean and variance of each feature simultaneously thereby avoiding a separate pass for the mean and variance calculation [52].

The processed dataset is made up of 11680 audio recordings which are split into three independent subsets with the split shown in table 6.1. The training set, is used to optimise the model parameters, hyperparameter tuning and architecture decisions are based on results reported on the held-out validation set, and the results are presented for the samples in the test set.

Table 6.1 Confidence score estimation dataset split.

Data subset	**Audio Recordings**
Training set	9285
Validation set	1184
Test set	1211
Total	11680

6.2 Training Procedure

Stochastic Gradient Descent (SGD) with Momentum is used to optimise either one of the objective function introduced in section 5.4. SGD tends to converge towards a local minimum quickly, but since each update is not guaranteed to be made on a representative subset of the underlying distribution, training may suffer from noisy weight updates [53]. The loss for a model which produces binary class predictions from a uniform distribution is expected to be 0.693. Surpassing this mark is an indication that the model is learning something useful.

The decision to use an online learning approach, such as SGD, is guided by the fact that for lattice and confusion network structures, each sample has a unique graphical structure over which the forward and backward passes must operate. This means that in order to take advantage of the performance gains of GPU processing, a padding structure must be adopted similar to that used by traditional sequence models. Due to the vast variability in depth and breadth of lattice and confusion network structures, the computational cost of the increased sequences is not compensated for by the computational gain of batch processing on a GPU [1].

Asynchronous Updates

In order to offset the limitation of training on CPUs, `Hogwild!`, a methodology which allows multiple processors access to run in parallel without the overhead of locking each process, is used. This means that each process has equal rights to access the same memory. As a result of this lock-free approach, each of the parallel SGD processes may overwrite memory being used by another parallel process. This concern is less critical when each SGD process is only updating a relatively small number of parameters [54].

Chapter 7

Experiments and Discussion

This chapter sets about describing and presenting the quantitative results of this work. A description of the graphemic Georgian ASR system as well as the metrics used for evaluating confidence scores is provided. The experimental results begin by looking at the experiments run on one-best sequences. A key question to address is whether introducing grapheme level features has a positive effect on confidence estimation and which grapheme merging schemes are effective. This is followed up with confusion network experiments which, in addition to investigating the impact of the competing hypotheses on the ability to predict confidence scores, present empirical findings for the introduction of grapheme features. All results presented in this chapter are evaluated on the test set unless indicated otherwise.

7.1 Experimental Setup

A set of hypotheses from an ASR system are required in order to train the confidence score models discussed in section 5. Additionally, a set of metrics with which to evaluate the confidence score performance is required. These two components are discussed before moving onto the experimental results.

7.1.1 Georgian ASR System

In order to facilitate the investigation of deep learning for confidence scores, it is beneficial for the ASR system to have a reasonable balance between correct and incorrect predictions. Many standard training approaches treat all types of error equally, therefore an underlying bias in the number of positive and negative samples can skew the reported cost. The LatticeRNN model is trained using binary cross entropy, which is susceptible to this problem. A moderate WER is desirable in order to avoid the problem of class imbalance.

The size of the Georgian speech corpus provided by the BABEL project [9] and the respective subsets is presented in table 7.1. This corpus is small enough to allow for iterative experimentation whilst at the same time being large enough to present the subtleties and complications associated with large vocabulary speech applications.

Table 7.1 Statistics for the content of the Georgian corpus.

Data subset	Recording duration (hours)	Number of utterances (thousands)	Number of words (thousands)
Training set	17.5	20.6	103.6
Test set	7.5	8.7	45.3

The WER of the CUED graphemic system for Georgian is about 30% [26]. Although this is still an imbalanced distribution of correct and incorrect arcs, it is not extreme enough to significantly hinder the confidence score investigation. A detailed breakdown for the different error types and their respective scores are provided in table 7.2.

Table 7.2 CUED Georgian ASR system performance.

Error type	Training set (%)	Test set(%)
Correct	69.1	72.5
Substitution	24.2	21.8
Deletion	6.7	5.7
Insertion	5.4	5.3
Word Accuracy	63.7	67.2

7.1.2 Evaluating Confidence

A reliable measure which is indicative of the ability of a model to predict confidence is critical for the development and comparative assessment of different solutions. In this work, two metrics for confidence are utilised, Normalised Cross Entropy (NCE) and the Area Under the Curve (AUC).

Normalised Cross Entropy

For a sequence of confidence predictions $\mathbf{c} = \{c_1, ..., c_T\}$ in a T length sequence, an empirical estimate of the ASR correctness is given by equation 7.1.

$$P(C=1) = \frac{1}{T}\sum_{t=1}^{T} c_t \tag{7.1}$$

The binary nature of the idealised confidence score allows the empirical estimate of incorrectness to be calculated by simply using the compliment: $P(C=0) = 1 - P(C=1)$. The binary cross entropy averaged over the T length sequence between the empirical estimate above and the reference confidence tags, given by $\bar{\mathbf{c}} = \{\bar{c}_1, \ ..., \ \bar{c}_T\}$, is simply

$$\overline{H}(\bar{\mathbf{c}}) = -\frac{1}{T}\sum_{t=1}^{T} \bar{c}_t \log(P(C=1)) + (1-\bar{c}_t)\log(1-P(C=1)) \tag{7.2}$$

The average binary cross entropy between the hypothesis generated by the ASR system and the reference confidence tags is defined by equation 7.3.

$$H(\mathbf{c}|\bar{\mathbf{c}}) = -\frac{1}{T}\sum_{t=1}^{T} \bar{c}_t \log(c_t) + (1-\bar{c}_t)\log(1-c_t) \tag{7.3}$$

By arranging equations 7.2 and 7.3 as shown in equation 7.4, the change in cross entropy after replacing the empirical estimate of the ASR system accuracy with the confidence score predictions can be measured. This is the definition of normalised cross entropy (NCE) which is a common metric for evaluating the accuracy of confidence score predictions [1, 55]. The aim is to obtain the highest possible NCE, where a positive NCE indicates that hypothesis confidence scores are more accurate than the empirical ASR estimate [31].

$$\text{NCE}(\mathbf{c}, \bar{\mathbf{c}}) = \frac{\overline{H}(\bar{\mathbf{c}}) - H(\mathbf{c}|\bar{\mathbf{c}})}{\overline{H}(\bar{\mathbf{c}})} \tag{7.4}$$

Area Under the Curve

A model which provides perfect scores, according to equation 5.18, is not required for optimal operation of downstream and upstream applications. The rank order of confidence score predictions is often sufficient. Consequently, the Area Under the Curve (AUC) metric is widely used in conjunction with the precision-recall curve for confidence scores. The precision-recall curve is a plot of the precision against the recall for various thresholds, where the precision and recall are given by equations 7.5 and 7.6 respectively.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{7.5}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{7.6}$$

TP, FP, and FN represent the number of true positive, false positive, and false negative samples respectively. This curve illustrates the trade off between a model which is too sensitive and accepts incorrect predictions and a model which is too conservative and may reject correct predictions. The AUC of this curve is an indicator of the discriminatory power of the model, therefore the goal is to force the precision-recall curve into the top right corner in order to maximise the total AUC score.

7.2 Experiments for One-Best Sequences

As discussed in chapter 3, word posterior probabilities, $p(w_i|\ \mathbf{O}, \lambda)$, are the simplest form of word-based confidence score for ASR. Typically, word posteriors tend to overestimate the confidence, in order to combat this the decision tree mapping, developed in [26], is applied to the word word posteriors.

Table 7.3 provides a comparison between the NCE and AUC results obtained before and after applying the decision tree mapping. There is a significant increase in the NCE score after mapping, however the AUC score is unmoved. This behaviour is expected since the the overestimated word posteriors are mapped closer to the true confidence thereby resulting in an improvement in overall NCE. Furthermore, the increasing monotonic nature of the mapping has no effect on the rank ordering of predictions, therefore the AUC is expected to remain the same.

Table 7.3 NCE and AUC scores for word posterior scores before and after applying decision tree mapping to raw word posteriors from one-best sequences.

Model	NCE	AUC
Unmapped word posteriors	-0.1978	0.9081
Decision tree mapped word posteriors	0.2755	0.9081

Predicting confidences by only using the word posteriors does not take advantage of the extensive confidences features explored in section 3.3. To remedy this, a BiLSTM model is trained on the one-best sequences using a number of word level features. These include a 50-dimensional word embedding, w_i, as described in section 5.3.1, the word duration in seconds, t_i^d, as well as the word posterior, $p(w_i|\ \mathbf{O}, \lambda)$. Two separate models are trained for mapped and unmapped word posteriors.

The most obvious contrast from the results presented in table 7.4, which compares the two BiLSTM models to their respective word posterior baselines, is that the relative increase in NCE provided by the BiLSTM is far greater for unmapped posteriors than for mapped posteriors. Specifically, the NCE increases from -0.1978 to 0.2765 for unmapped posteriors

compared to the increase from 0.2755 to 0.2911 for mapped confidence scores. This is because the BiLSTM structure has the scope to implicitly learn the decision tree mapping. Although one may expect the BiLSTM model to completely learn the simple linear mapping, parameter initialisation may have limited the ability of the unmapped BiLSTM to completely match the results obtained by the mapped BiLSTM. This is a possible avenue for investigation in future work. The BiLSTM using mapped word posteriors provides a strong improvement in NCE and AUC scores. For this reason, all results going forward are reported for mapped confidence scores only.

Table 7.4 Implementing a BiLSTM on top of mapped or unmapped arc posteriors.

	Mapped		**Unmapped**	
Model	**NCE**	**AUC**	**NCE**	**AUC**
Word posteriors	0.2755	0.9081	-0.1978	0.9081
BiLSTM	**0.2911**	**0.9121**	0.2765	0.9033

The learning curve for the BiLSTM using the mapped word posteriors, in figure 7.1, gives an indication of the optimisation process by plotting the train and validation losses over all epochs. The gap between the validation and training curves from about halfway through the training procedure indicates that the model begins to overfit the data to some degree, however this is not a cause for concern when one considers the resolution of the y-axis.

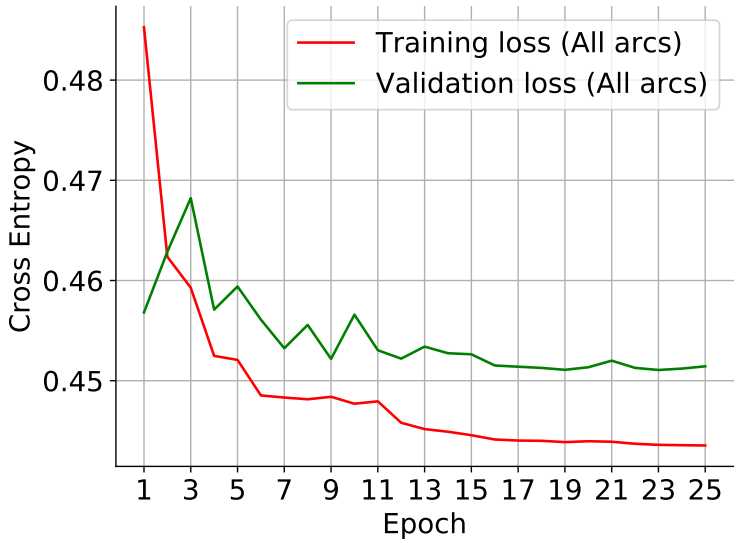


Fig. 7.1 Learning curve for the BiLSTM using mapped word posteriors.

As discussed in section 3.3, sub-word level features, in particular grapheme level features, can be a useful for predicting confidence scores. These features must be merged into

a fixed length representation for easy computation. To this end, a number of attention mechanisms are applied directly over the grapheme feature sequence to generate a compressed representation. The best performing model from the previous experiment, the BiLSTM with mapped posteriors, is presented in the first row of table 7.5. This is compared against 'flat' multiplicative, scaled dot product, and additive self-attention schemes as defined in section 5.3.2. Although the additive attention mechanism yields the highest average score, there is very little difference between the three forms of self-attention presented.

Table 7.5 Comparing different self-attention methods over the grapheme sequence

Attention Type	NCE	AUC
-	0.2911	0.9121
Multiplicative	0.2942	0.9129
Scaled Dot Product	0.2940	0.9125
Additive	**0.2944**	**0.9129**

The gains obtained from the introduction of grapheme features warrant further experimentation. One component to investigate is whether introducing more information into the key leads to better fixed length grapheme representations. One possibility is to define the key such that it incorporates word duration by concatenating it with the grapheme feature vector. This form of attention mechanism, as described in detail in section 5.3.2, tries to associate the word duration with the respective grapheme durations in the grapheme feature vector. Experimental results presented in table 7.6 indicate that the word duration does not provide any gains compared to the additive self-attention model. Something to note is that replacing the grapheme duration with the word duration produces very similar NCE and AUC scores. This finding indicates that a duration indication together with the grapheme embeddings provide a strong basis for an effective key, but that the grapheme and word duration are not compatible features.

Table 7.6 Investigating different key configurations.

Additive Attention Key	NCE	AUC
$[\,\mathbf{s}_i^{(j)}\,]^T$	0.2929	0.9125
$[\,d_i^{(j)}]^T$	0.2920	0.9128
$[\,\mathbf{s}_i^{(j)}\;d_i^{(j)}\,]^T$	**0.2944**	**0.9129**
$[\,\mathbf{s}_i^{(j)}\;t_i^{(d)}\,]^T$	0.2944	0.9128
$[\,\mathbf{s}_i^{(j)}\;d_i^{(j)}\;t_i^{(d)}\,]^T$	0.2928	0.9188

The next set of experiments serves a dual purpose. The first objective is to determine whether the 'flat' attention scheme, which operates directly over the grapheme sequence, can be improved by adding a grapheme encoder and attending over the encoder hidden states. The second objective is to obtain some insight as to what type of bidirectional recurrent structure performs best at this task. Table 7.7 provides a comparison of the BiRNN, BiGRU, and BiLSTM encoders with additive attention applied on the hidden state sequence for each of these encoders. The results indicate that the BiGRU encoder provides a significant improvement as reflected in the NCE and AUC score. This is not surprising since as described in chapter 4, the GRU was developed as an improvement on the already existing LSTM. Furthermore, this experiment indicates that the BiGRU grapheme encoder supersedes the best 'flat' attention system. This is unsurprising since the BiGRU is able to generate a latent representation for the grapheme sequence which is optimised towards exposing the sequential characteristics that are pertinent to confidence score estimates.

Table 7.7 Comparing BiRNN, BiGRU, and BiLSTM grapheme encoders using additive self-attention over the encoder hidden states.

Encoder Type	**NCE**	**AUC**
-	0.2944	0.9129
BiRNN	0.2938	0.9135
BiGRU	**0.2978**	**0.9139**
BiLSTM	0.2947	0.9123

The learning curve for the BiLSTM model which incorporates grapheme information via a BiGRU grapheme encoder with additive attention is presented in figure 7.2. The training and validation losses converge to 0.44 and 0.45 respectively, which indicates that the model is able to generalise reasonably well. Except for the region around the fifth epoch, the training and validation loss in this plot makes a smoother transition to convergence than observed in figure 7.1.

As described in section 2.4.3, the method for generating the one-best sequences used in this work does not allow the LM and AM scores to be maintained. In spite of this fact, one can attempt to recover an estimate for this information by matching each arc in the one-best sequence with an arc in the corresponding lattice as described in section 5.3.4. Table 7.8 presents a listing of the number of one-best arcs in each dataset which could not be matched to a lattice arc. Since none of these are from the test set, the three offending one-best sequences are simply dropped from the training and validation process.

The approximate AM and LM scores do not compliment the existing features and as a result, are not helpful for confidence estimation. This is clearly evident from the NCE

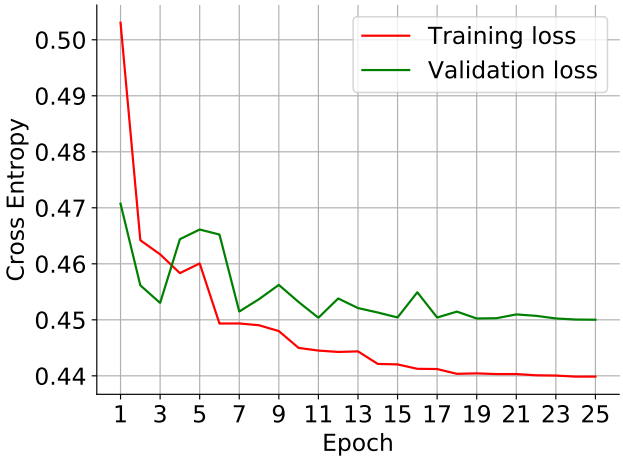


Fig. 7.2 Learning curve for the BiLSTM with a BiGRU grapheme encoder with additive attention.

Table 7.8 The number of arcs in the one-best dataset which could not be matched to an arc in the corresponding lattice.

Subset	Unmatched arcs	Effected one-best sequences
Train	1	1
Validation	2	2
Test	0	0
Total	**3**	**3**

and AUC scores given in table 7.9. Exact AM and LM scores are usually useful confidence estimation features, therefore it is suspected that the estimates obtained from the arc matching process introduce noise into the existing feature set.

Table 7.9 The impact of introducing estimated AM and LM scores for one-best sequences.

Additional Features	NCE	AUC
-	**0.2978**	**0.9139**
approximate AM and LM	0.2942	0.9131

The precision-recall curve, presented in figure 7.3, is used to assess the relative performance of the enhanced BiLSTM model against mapped posteriors. The most significant gains in precision are obtained for low recall levels.

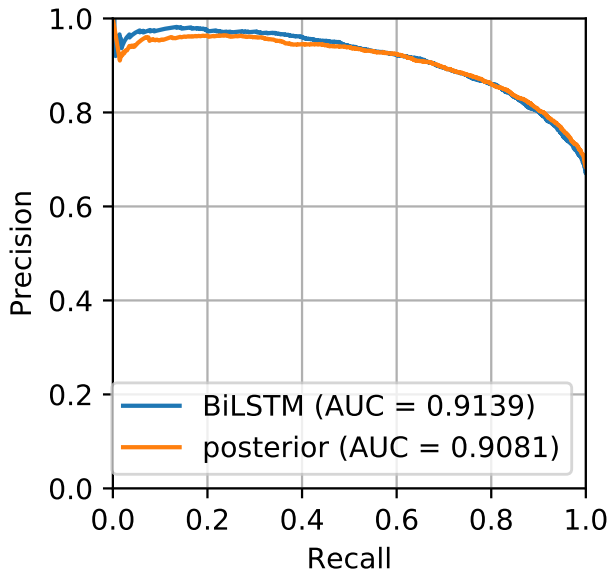


Fig. 7.3 Precision-recall curve for the BiLSTM with a BiGRU grapheme encoder with additive attention and the mapped arc posteriors.

7.3 Experiments for Confusion Networks

The BiLSTM model developed for the one-best sequences in the preceding section is a special case of the LatticeRNN model, presented in chapter 5, where the arc merging mechanism is not required. In this section, the ability of competing arcs to provide additional information for predicting confidence on the one-best sequence is explored. Additionally, experiments are devised to determine whether the gains observed from introducing grapheme information in one-best sequences transfer to confusion networks.

The first step is to evaluate the performance of the word-based confusion network model against the word-based one-best sequence model. These results are shown in table 7.10. The LatticeRNN model can be trained with the objective of minimising the loss over the one-best sequence or minimising the loss over all arcs in the confusion network. The results for these two modes are shown in rows two and three of table 7.10. Unsurprisngly, the AUC for the one-best sequence is highest when the confusion network is optimised with respect to the one-best sequence only. That being said, a minor improvement in the NCE is observed when the confusion network is optimised for all arcs. This indicates that there is some advantage to propagating gradients through all arcs during training even if the optimisation is limited to a subset of these arcs.

As discussed in the chapters leading up to this section, the competing hypotheses in lattices and confusion networks are valuable in their own right for upstream and down-

Table 7.10 Evaluation of the one-best sequence model against the confusion network system for a word-based system.

Input structure type	Loss	NCE	AUC
One-best	One-best	0.2911	0.9121
Confusion Network	One-best	0.2931	0.9193
Confusion Network	Confusion Network	0.2934	0.9178

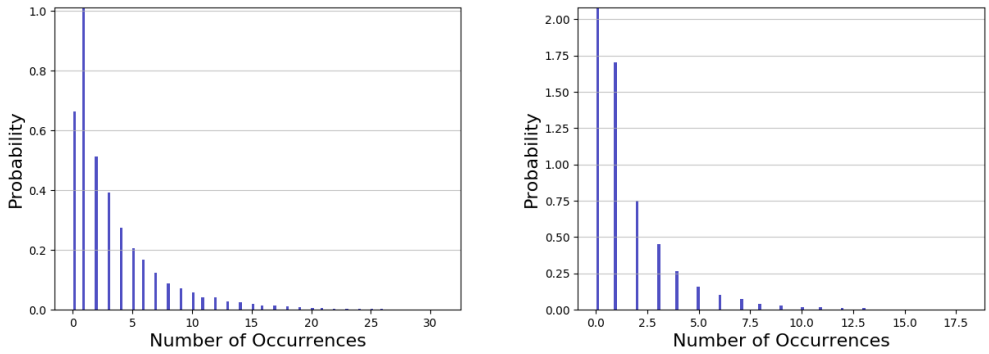
stream applications. As a result, it is useful to determine the confidence associated with these competing hypotheses as presented in table 7.11. The NCE metric indicates that the LatticeRNN model is drastically better at predicting confidence scores for the alternative hypotheses than the one-best arc. A potential explanation for this is that since there are many more competing arcs than in the one-best sequence, the model is expected to have a heavy bias towards learning behaviour which allows it to most closely match predictions in the competing hypotheses. This is a consequence of the model treating each arc with equal importance as defined in the loss function in section 5.4.

Table 7.11 Performance difference between the one-best sequence and all arcs in the word-based confusion network optimised on all the arcs.

Evaluation Type	NCE	AUC
One-best arcs	0.2934	0.9178
All arcs	0.4962	0.8356

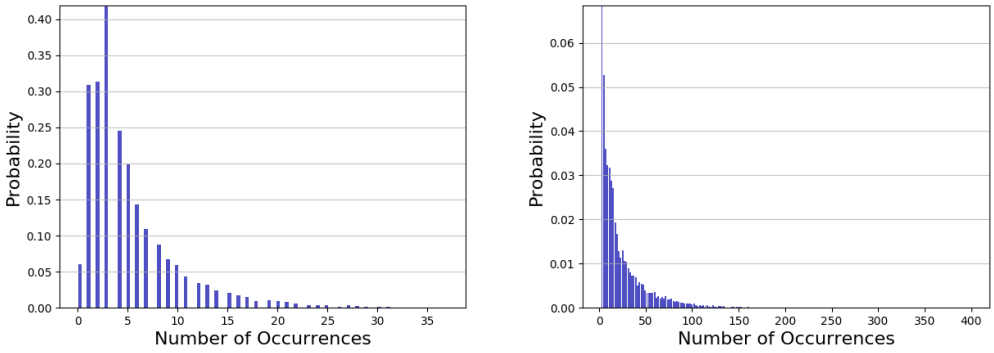
The AUC reported in table 7.11 may seem to contradict this conjecture at first, since the AUC score drops when evaluating all arcs in the confusion network, however the increased complexity involved in rank ordering all arcs in the confusion network, which are far more numerous than the number of one-best sequence arcs, means that the AUC metric is expected to drop even though the predictive ability of the model has improved.

Furthermore, a difference in the distribution of classes in the one-best sequence arcs compared to the full confusion network could exacerbate this behaviour. Figures 7.4 and 7.5 demonstrate the class distribution by presenting the distribution of correct and incorrect predictions in the one-best sequences and confusion networks respectively. For the one-best sequences, there tend to be more correctly predicted arcs than incorrectly predicted arcs. This is expected since this is the best prediction as ranked by ASR system. This contrasts the distributions for the full confusion network where the number of incorrect arcs is far greater than the number of correct arcs. These two sets of distributions describe significantly different behaviour, therefore the difference in AUC is to be expected. Consequently, this optimisation strategy may result in the model being susceptible to understating arc confidence.



(a) Correctly predicted arcs (b) Incorrectly predicted arcs

Fig. 7.4 The empirical distributions of the number of occurrences of the two classes in the one-best arcs in the confusion network dataset.



(a) Correctly predicted arcs (b) Incorrectly predicted arcs

Fig. 7.5 The empirical distributions of the number of occurrences of the two classes in the confusion network dataset.

Following on from the one-best experiments, the effect of additional information, in terms of graphemic features as well as the word level AM and LM scores, is investigated. The same arc matching procedure as for one-best sequences is carried out in order to insert approximate AM and LM scores into the confusion networks. Unlike for the one-best sequences, which were force aligned to obtain the exact grapheme alignment, approximate grapheme features must be obtained from the grapheme-marked lattice using the arc matching mechanism. The increased structural complexities of the confusion networks means that the likelihood of being unable to find a grapheme-marked lattice arc for all arcs in a confusion network is reduced. This is evident from the results in table 7.12 which present the number of confusion networks per-subset which have an unmatched arc and the number of unmatched arcs. As was the case with the one-best sequences, unmatched confusion networks are removed from

the dataset, however in this scenario 25 confusion networks are removed from the test set. This modification has a tangible effect on the previously reported test results, therefore all 199 effected confusion networks are removed from the word-based confusion network dataset and the metrics are recomputed.

Table 7.12 The number of arcs in the confusion network subset which could not be matched to an arc in the corresponding grapheme-marked lattice.

Subset	Unmatched arcs	Effected confusion networks
Train	212	146
Validation	42	28
Test	31	25
Total	**285**	**199**

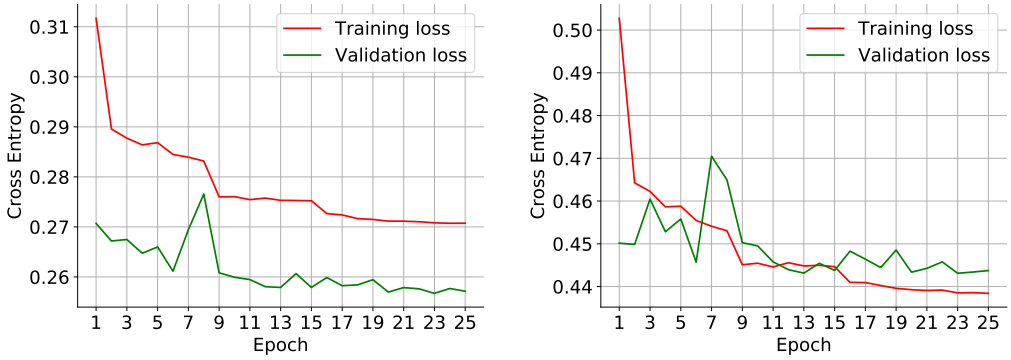
Table 7.13 provides a comparison between the word-based LatticeRNN model, and the LatticeRNN model which incorporates grapheme features. On the back of the results obtained for one-best sequences, the same BiGRU grapheme encoder with additive attention is used for the grapheme-marked confusion networks. Performance gains in terms of NCE and AUC are observed after the introduction of grapheme features. This indicates that the arc matched grapheme features are accurate enough to provide complimentary predictive ability to the LatticeRNN model. The relative gains by introducing grapheme information into confusion networks surpasses the gains seen in the one-best sequences. This is further evidence of the predictive power of sub-word level features for confidence estimation as well as the validity of the arc matching algorithm as a mechanism for introducing additional arc-level information into confusion networks. The introduction of the AM and LM estimates reveals minor improvements in NCE and AUC performance. Although this is better than the performance degradation observed for one-best sequences, the statistical significance of these findings are questionable.

Table 7.13 Confidence score metrics for the LatticeRNN models trained with respect to all arcs in the confusion network

	One-best arcs		All arcs	
Features	**NCE**	**AUC**	**NCE**	**AUC**
Word-based	0.2934	0.9201	0.4959	0.8406
+ approximate grapheme features	0.2998	0.9228	0.4993	0.8432
+ approximate LM and AM	0.3004	0.9231	0.5013	0.8444

The learning curves in figure 7.6 presents the evolution of training and validation losses for LatticeRNN with approximate grapheme features for all arcs as well as for the one-best

sequence of arcs. For figure 7.6a, the validation loss is lower than the training loss throughout the optimisation process. This is not typical behaviour, and it indicates that the validation set is not representative of the difficulty of the training set. This behaviour is not as defined for the one-best sequence of arcs and by the end of training, the validation loss is higher than the training loss, as expected. Further investigations into the relative sentence lengths and the number of competing arcs for each subset is required in order to pinpoint whether this is simply a consequence of the confusion networks in each subset. Furthermore, at about the eighth epoch mark the validation loss spikes. This is indicates that the model parameters overfit the data momentarily during training.



(a) All arcs

(b) One-best arcs

Fig. 7.6 The learning curves for the LatticeRNN model with BiGRU encoder with additive attention for the approximate grapheme features.

Figure 7.7 presents the precision-recall curve for the LatticeRNN model with approximate grapheme features against the mapped word posteriors. In a similar way to the BiLSTM model, the gains over the mapped posteriors are made in lower recall region.

7.4 Discussion

These experiments present an incremental process in which the complexity of the confidence score estimation model is built up from a single feature estimator to an end-to-end deep learning model which uses several features and operates on a complicated graphical structure. Together with this increasing complexity, increasingly accurate confidence scores are observed. The contribution of grapheme features and the investigation of various methods for introducing this information for a real-word dataset provides significant gains to system performance. Experimenting on confusion networks provided scope for the proposed grapheme feature merging technique to be evaluated on a graph structure.

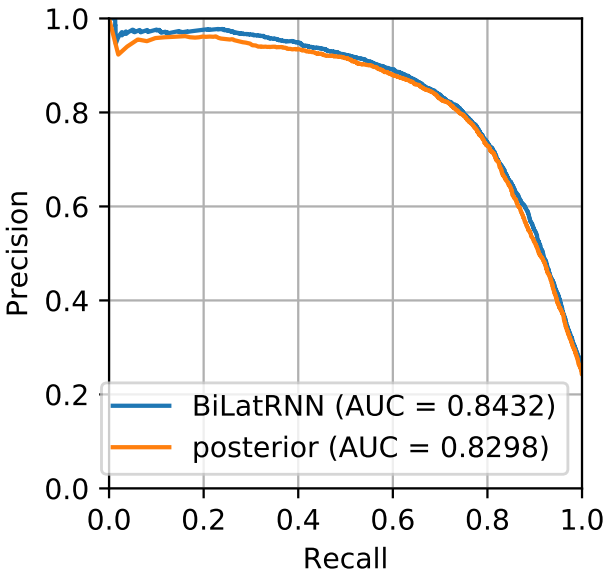


Fig. 7.7 Precision-recall curve for all confusion network arcs for the BiLatticeRNN model with the approximate grapheme features information and mapped arc posteriors.

Unlike confusion networks, which are often used to yield one-best transcriptions[1], lattices are commonly used for downstream applications such as information retrieval and upstream application such as speaker adaptation of acoustic models. Thus it is useful to investigate improvements in confidence score estimation for lattices in the context of those applications. For this reason, as well as the time constraints associated with this work, lattice experiments were not presented.

[1] Although confusion networks can be used for downstream tasks such as information retrieval, the use of lattices is more common.

Chapter 8

Conclusion

This chapter highlights the main themes of this work on confidence scores for sequence data. The major findings are presented in summary followed by suggestions for future work.

8.1 Summary

The importance of confidence scores for both downstream and upstream applications in speech processing is rapidly increasing as speech activated devices continue to become increasingly involved in everyday human interactions. A flexible framework which can be generated with minimal expert knowledge is favourable. Rather than predicting confidence on a traditional one-best word sequence, graph-like structures, such as lattices and confusion networks, are favoured as they provide additional information about competing hypotheses. This additional information from the competing hypotheses is valuable for confidence estimation. Furthermore, grapheme features contain significant predictive power for confidence estimation. To this end, an end-to-end deep learning model for introducing grapheme features into the existing LatticeRNN framework was proposed.

The empirical experimentation conducted in this work shows that a grapheme encoder, constructed by a bidirectional GRU with an additive self-attention mechanism operating over the hidden states, is able to effectively restrict the grapheme information to a fixed form. Introducing this fixed form representation into the LatticeRNN model saw significant improvements in NCE and AUC scores for one-best sequences as well as for confusion networks. Furthermore, an arc matching scheme for extracting estimated features, such as graphemes or LM and AM model scores, is proposed.

8.2 Future work

Possible improvements to the proposed model could involve moving away from fixed word and grapheme embeddings and having the model learn the embedding during training. Not only could this improve confidence score estimation, but analysing the grapheme clusters before and after training could provide the model with a degree of interpretability. Furthermore, much of the results presented in this work are preliminary in the sense that an extensive hyperparameter search could yield significant improvements without modification to the architecture. Together with this, additional features could be investigated such as acoustic stability through the lens of a perturbation analysis. Future work may wish to experiment on different datasets to verify that LatticeRNN and the extensions introduced in this work are able to generalise. Finally, UAPS pruning could be applied to the lattices rather than the beam pruning strategy used in this work. This would reduce the I/O overhead required to load each lattice during processing and the number of arcs to traverse during training.

References

[1] Q. Li, P. Ness, A. Ragni, and M. Gales, "Bi-directional lattice recurrent neural networks for confidence estimation," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 6755–6759.

[2] L. Uebel and P. Woodland, "Speaker adaptation using lattice-based mllr," in *ISCA Tutorial and Research Workshop (ITRW) on Adaptation Methods for Speech Recognition*, 2001.

[3] H. Y. Chan and P. Woodland, "Improving broadcast news transcription by lightly supervised discriminative training," in *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 1. IEEE, 2004, pp. I–737.

[4] F. Wessel, R. Schluter, K. Macherey, and H. Ney, "Confidence measures for large vocabulary continuous speech recognition," *IEEE Transactions on Speech and Audio Processing*, vol. 9, no. 3, pp. 288–298, March 2001.

[5] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. Odell, D. Ollason, D. Povey *et al.*, "The htk book," *Cambridge university engineering department*, vol. 3, p. 175, 2002.

[6] F. L. Kreyssig, C. Zhang, and P. C. Woodland, "Improved tdnns using deep kernels and frequency dependent grid-rnns," *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Apr 2018. [Online]. Available: http://dx.doi.org/10.1109/icassp.2018.8462523

[7] M. Adda-Decker and L. Lamel, *The Use of Lexica in Automatic Speech Recognition*. Dordrecht: Springer Netherlands, 2000, pp. 235–266. [Online]. Available: https://doi.org/10.1007/978-94-010-9458-0_8

[8] K. Knill, M. Gales, K. Kyriakopoulos, A. Ragni, and Y. Wang, "Use of graphemic lexicons for spoken language assessment," 2017.

[9] M. J. F. Gales, K. M. Knill, and A. Ragni, "Low-resource speech recognition and keyword-spotting," in *Speech and Computer*, A. Karpov, R. Potapova, and I. Mporas, Eds. Cham: Springer International Publishing, 2017, pp. 3–19.

[10] M. Gales, S. Young *et al.*, "The application of hidden markov models in speech recognition," *Foundations and Trends® in Signal Processing*, vol. 1, no. 3, pp. 195–304, 2008.

[11] G. Evermann and P. C. Woodland, "Large vocabulary decoding and confidence estimation using word posterior probabilities," in *2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.00CH37100)*, vol. 3, June 2000, pp. 1655–1658 vol.3.

[12] S. Kumar and W. Byrne, "Minimum bayes-risk decoding for statistical machine translation," JOHNS HOPKINS UNIV BALTIMORE MD CENTER FOR LANGUAGE AND SPEECH PROCESSING (CLSP), Tech. Rep., 2004.

[13] L. Mangu, E. Brill, and A. Stolcke, "Finding consensus among words: Lattice-based word error minimization," in *Sixth European Conference on Speech Communication and Technology*, 1999.

[14] R. San-Segundo, B. Pellom, K. Hacioglu, W. Ward, and J. M. Pardo, "Confidence measures for spoken dialogue systems," in *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 01CH37221)*, vol. 1. IEEE, 2001, pp. 393–396.

[15] M. S. Seigel and P. C. Woodland, "Combining information sources for confidence estimation with crf models," in *Twelfth Annual Conference of the International Speech Communication Association*, 2011.

[16] H. Jiang, "Confidence measures for speech recognition: A survey," *Speech communication*, vol. 45, no. 4, pp. 455–470, 2005.

[17] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, Feb 1989.

[18] C. V. Neti, S. Roukos, and E. Eide, "Word-based confidence measures as a guide for stack search in speech recognition," in *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2. IEEE, 1997, pp. 883–886.

[19] G. Evermann and P. Woodland, "Posterior probability decoding, confidence estimation and system combination," 2000.

[20] J. Pinto and R. N. V. Sitaram, "Confidence measures in speech recognition based on probability distribution of likelihoods," in *INTERSPEECH*, 2005.

[21] T. Zeppenfeld, M. Finke, K. Ries, M. Westphal, and A. Waibel, "Recognition of conversational telephone speech using the janus speech engine," in *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 3, April 1997, pp. 1815–1818 vol.3.

[22] G. Williams and S. Renals, "Confidence measures from local posterior probability estimates," *Computer Speech & Language*, vol. 13, no. 4, pp. 395–411, 1999.

[23] M. Weintraub, F. Beaufays, Z. Rivlin, Y. Konig, and A. Stolcke, "Neural-network based measures of confidence for word recognition," in *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, April 1997, pp. 887–890 vol.2.

[24] A. Sanchis, A. Juan, and E. Vidal, "Estimating confidence measures for speech recognition verification using a~smoothed naive bayes model," in *Pattern Recognition and Image Analysis*, F. J. Perales, A. J. C. Campilho, N. P. de la Blanca, and A. Sanfeliu, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 910–918.

[25] T. Kemp and T. Schaaf, "Estimating confidence using word lattices," in *in Proceedings of EuroSpeech*, 1997, pp. 827–830.

[26] Q. Li, "Confidence scores for speech processing," 2018.

[27] T. J. Hazen, S. Seneff, and J. Polifroni, "Recognition confidence scoring and its use in speech understanding systems," *Computer Speech & Language*, vol. 16, no. 1, pp. 49–67, 2002.

[28] J. Lafferty, A. McCallum, and F. C. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," 2001.

[29] D. Yu, L. Deng, and A. Acero, "Hidden conditional random field with distribution constraints for phone classification," in *Tenth Annual Conference of the International Speech Communication Association*, 2009.

[30] M. S. Seigel and P. C. Woodland, "Detecting deletions in asr output," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2014, pp. 2302–2306.

[31] A. Ragni, Q. Li, M. J. Gales, and Y. Wang, "Confidence estimation and deletion prediction using bidirectional recurrent neural networks," in *2018 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2018, pp. 204–211.

[32] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018.

[33] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[34] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, April 1967.

[35] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[36] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct 1990.

[37] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[38] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[39] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2013. [Online]. Available: http://dx.doi.org/10.1109/ICASSP.2013.6638947

[40] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014.

[41] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with lstm recurrent networks," *J. Mach. Learn. Res.*, vol. 3, pp. 115–143, Mar. 2003. [Online]. Available: https://doi.org/10.1162/153244303768966139

[42] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[43] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015. [Online]. Available: http://dx.doi.org/10.18653/v1/D15-1166

[44] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014.

[45] G. Tang, M. Müller, A. Rios, and R. Sennrich, "Why self-attention? a targeted evaluation of neural machine translation architectures," 2018.

[46] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 3111–3119. [Online]. Available: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

[47] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, p. 135–146, Dec 2017. [Online]. Available: http://dx.doi.org/10.1162/tacl_a_00051

[48] F. Ladhak, A. Gandhe, M. Dreyer, L. Mathias, A. Rastrow, and B. Hoffmeister, "Latticernn: Recurrent neural networks over lattices." in *INTERSPEECH*, 2016, pp. 695–699.

[49] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[50] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[51] T. E. Oliphant, *A guide to NumPy*. Trelgol Publishing USA, 2006, vol. 1.

[52] B. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962.

[53] S. Ruder, "An overview of gradient descent optimization algorithms," 2016.

[54] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in neural information processing systems*, 2011, pp. 693–701.

[55] T. Schaaf and T. Kemp, "Confidence measures for spontaneous speech recognition," in *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, April 1997, pp. 875–878 vol.2.