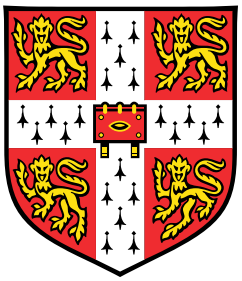
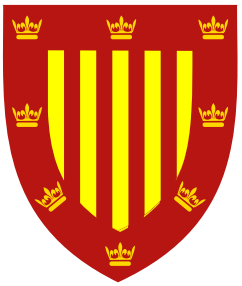
Confidence Scores for Speech Processing





Qiujia Li

Supervisor: Prof. Mark Gales

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Engineering

Peterhouse May 2018

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 12,000 words including footnotes, figures, tables and equations.

Qiujia Li
May 2018

Acknowledgements

First, I would like to thank my supervisor, Prof. Mark Gales, for his guidance and patience throughout my fourth-year project. The weekly meetings never failed to enlighten me. Not only did I learn knowledge about the subject matter, but also the approaches to addressing challenging problems. During the entire project, Dr Anton Ragni has also provided valuable insights and great hands-on support. I also would like to thank Dr Yu Wang and Prof. Phil Woodland for useful discussions on multiple encounters.

I really appreciate the help from my friends Chengkai Zhang, and Xinyu Wang on several implementation issues and proofreading of this report. Finally, I would like to thank my parents and my friends at Cambridge, who made every moment of my undergraduate life memorable and enjoyable.

Abstract

In many applications involving speech recognition technology, it is a common case that apart from the transcription itself, an indication of reliability associated with the transcription is needed for making downstream decisions. The confidence score provides such a measure and it is estimated by using intermediate information available during the recognition process. In this report, the fundamentals of speech recognition are firstly reviewed, especially the decoding stage where lattices and confusion networks that encode a rich amount of meta information are produced. Based on the understanding of information sources, current approaches for confidence score estimation are briefly introduced, including the features and models that combine these features.

Despite the effectiveness of some of these approaches, few of them are ideal. In automatic speech recognition, the recogniser could produce results in the form of one-best sequences, confusion networks and lattices with an increasing amount of information and complexity. The system could also operate at the sub-word level, the word level or the utterance level. Therefore, a model that functions like a classifier on inputs with various structures and levels of complexity, that could incorporate an arbitrary number and type of input features, that could be applied at different levels of granularities, and that could extract useful information from the structured data would be perfect. Drawing from the recent advancement in deep learning methods, such as the bidirectional long short-term memory network and the attention mechanism, a variant of the recurrent neural network – LATTICERNN – is introduced. LATTICERNN integrates recurrent neural network with the idea of the forward-backward algorithm that runs on lattices. The major benefit of the LATTICERNN model is the generality and flexibility it offers as a framework.

Building upon knowledge about speech recogniser and deep learning models, the theory of the LATTICERNN model is formulated. Implementation details are also described on account of the special architecture of the model, which also include the approaches to labelling all arcs on the lattices or the confusion networks for training. It is widely acknowledged that achieving efficient training of neural networks on non-uniform inputs such as sequences is challenging, let alone the more complex structures such as graphs for LATTICERNN . By carefully distributing the training task on multiple cores of a machine, the learning is greatly accelerated. Subsequent

experiments are conducted in a bottom-up fashion where the complexity of the network progressively increases. Experimental results show that LATTICERNN works well for one-best sequences, but its performance on confusion networks is under expectation. The degradation of performance is closely examined and experiments suggest that the arc tagging approach could be the bottleneck for further improvements using LATTICERNN .

LATTICERNN shows attractive properties for confidence score estimation. This project could be continued by detailed investigation on the arc tagging approach, and further experimentations on lattices with different datasets. Integrating the LATTICERNN model into other in-system applications relying on confidence scores would be a long-term objective. Nevertheless, this work demonstrates the working concepts of LATTICERNN and its potential to yield better performance in confidence score estimation.

Table of Contents

List of Figures

List of Tables

Notation

Confidence Scores

α	Forward log probability in the forward-backward algorithm
AM_e	The acoustic model score of an arc e
β	Backward log probability in the forward-backward algorithm
D_e	The destination/child node of an arc e
e	An arc in a graph
$\mathcal{E}$	A set of edges in a graph
e^i	An incoming edge of the node S_e
e^o	An outgoing edge of the node D_e
$\mathcal{G}$	A general graph
LM_e	The language model score of an arc e
N	A node in a graph
$\mathcal{N}$	A set of nodes in a graph
$\mathcal{Q}_e$	A set of paths in a graph that contains the edge e
S_e	The source/parent node of an arc e
t_e	End time of a word
$\tilde{p}$	Decision tree-mapped arc posterior probability of a confusion network
t_s	Start time of a word
$\mathbf{x}$	Input feature vector
y	Estimated confidence score

Deep Learning

$\mathbf{b}$	Bias vector
$\leftarrow$	Backward state vector
$\mathbf{c}$	Cell state in LSTMs
Δt	Word duration
$\mathbf{f}$	Forget gate in LSTM
$\mathcal{F}(\cdot)$	A non-linear function
$\rightarrow$	Forward state vector

$\mathbf{h}$	Hidden state in RNNs
$\mathbf{i}$	Input gate in LSTM
$\mathbf{k}$	Key vector of the attention mechanism
$\mathbf{k}$	The key vector of attention mechanism
$\mathcal{L}$	The loss function
y^*	The training target
LSTM$(\cdot)$	LSTM recurrent unit operation
$\mathbf{o}$	Output gate in LSTM
$\phi(\cdot)$	An activation function
$\sigma(\cdot)$	Sigmoid activation function
$\mathbf{U}$	Input weight matrix
$\mathbf{W}$	Recurrent weight matrix
$\mathbf{w}$	Word embedding

Indices and Sizes

d	Number of dimensionalities
j	Index of incoming arcs of a node
k	Index of outgoing arcs of a node
L	Total number of data sample
m	Total number of outgoing arcs of a node
n	Total number of incoming arcs of a node
T	Length of an sequence
t	Time index

Probabilities and Distributions

$\mathcal{H}$	Entropy
$P(\cdot)$	The probability mass function of a discrete random variable
$p(\cdot)$	The probability density function of a continuous random variable
$Z(\cdot)$	Normalisation or partition function

Speech Recognition

a_{ij}	The HMM transition probability from state i to state j
$b_j(\cdot)$	The HMM output distribution for state j
$\mathbf{q}$	A state sequence
$\boldsymbol{\theta}$	All parameters associated with HMMs
$\mathcal{Q}$	All possible state sequences
γ	The grammar scale factor
$\hat{\mathbf{q}}$	The most likely state sequence
$\hat{\mathbf{W}}$	The most likely word sequence

$\mathbf{O}$	An observation sequence
q	The hidden state variable at time t
ρ	The insertion penalty
s	A hidden state in HMMs
$\mathbf{W}$	An ASR hypothesis consisting of a sequence of words
w	A word in a word sequence

Acronyms / Abbreviations

AM	Acoustic Model
ANN	Artificial Neural Network
ASR	Automatic Speech Recognition
AUC	Area under Curve
BLSTM	Bidirectional Long Short-Term Momery
BPTS	Backpropagation Through Structure
BPTT	Backpropagation Through Time
CRF	Conditional Random Field
DAG	Directed Acyclic Graph
EM	Expectaion Maximisation
FPR	False Positive Rate
GMM	Gaussian Mixture Model
GSF	Grammar Scale Factor
HMM	Hidden Markov Model
LM	Language Model
LSTM	Long Short-Term Memory
MAP	Maximum a Posteriori
NCE	Normalised Cross Entropy
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
SGD	Stochastic Gradient Descent
TPR	True Positive Rate
UAPS	Unique Arcs per Second
WER	Word Error Rate

Chapter 1

Introduction

1.1 Motivation

Speech processing technology, specifically automatic speech recognition (ASR), enables us to communicate with machines more naturally comparing with traditional interfaces. Recently, personal voice assistants powered by ASR technology, such as Google Home, Amazon Alexa and Apple Siri, have gained significant popularity. Besides the consumer-based applications, large-scale transcription of speech is also highly demanded by some industries including television broadcasting and telephony. Strongly motivated by practical applications, the performance of automatic speech recognition (ASR) systems is consistently improving owing to the advancement of artificial neural networks (ANNs) [9]. Despite the state-of-the-art recognition accuracy, ASR systems sometimes produce erroneous transcriptions, especially for low-resource languages and speech signals under more realistic settings where channel distortions, speaker variations and ambient noise are significant. Generally, the transcriptions produced are always the best guess made by the recogniser. However, the transcriptions may not be equally good. In other words, the result is more likely to be correct if the mismatch between the testing and training conditions is minimal, which normally means that the speaker is native and talks in a quite open space. On the contrary, when the system is sub-optimal or the mismatch of acoustic conditions is prominent, the result would be much less reliable. To this end, confidence scores, as a measure of the reliability of the transcription, are highly demanded [32].

Confidence scores play a significant role in many downstream applications [25]. For example, it is sensible for the voice assistant to understand how confident it is about the transcription before executing the corresponding command asked by the user. If the confidence scores are relatively low, the assistant could offer better user experience if it confirms the message with the user to avoid misunderstanding.

In short, confidence scores are a highly desirable feature associated with the speech recogniser and could provide crucial information for decision making at some later stages in many systems that involve speech recognition technology. Improving the methodology for estimating confidence measures has been an open research problem in the field of speech processing. In this project, focus has been devoted to employing deep learning models and to creating a general framework for confidence scores.

1.2 Approach

Broadly speaking, this project aims to use intermediate information produced during the recognition process, to create appropriate models that could capture this information, and to efficiently train the models to infer the confidence scores.

First, standard approaches to generating confidence scores are reviewed. Secondly, a simple linear deep learning model is introduced for this task to verify that neural networks are capable of modelling the transform from input features to confidence scores and yielding state-of-the-art results. Thirdly, the more advanced yet more complex model is built upon the previous model and is expected to further improve the performance in confidence score estimation. This progressive approach allows the new techniques or models to be applied in an interpretable fashion, *i.e.* any factor that varies in each step is tractable.

1.3 Report Outline

This report contains 7 chapters and is structured as follows:

- Chapter 2 briefly covers the fundamentals of ASR systems, particularly the decoding process, which will be referred to throughout the report.
- Chapter 3 provides some technical concepts and details on how confidence scores are estimated in previous research work.
- Chapter 4, building upon Chapter 3, introduces the deep learning model named LATTICERNN as an attractive approach for confidence estimation.
- Chapter 5 addresses the computational challenges when implementing LATTICERNN.
- Chapter 6 presents the experiments carried out to verify the concept of LATTICERNN with some analysis of experimental results.
- Chapter 7 draws high-level conclusions based on both theory and practice accounted in the report. It also suggests future directions where this work could be useful.

Chapter 2

Automatic Speech Recognition

ASR is the process of converting the acoustic speech signal into its corresponding textual representation. It is considered to be a challenging task for real-life applications due to both inter-speaker variability (including physiological differences and pronunciation differences of accents or dialects) and intra-speaker variability (including different styles of speech). Other hardware and environmental factors such as channel distortion, reverberation and background noise pose more difficulties for achieving effective speech to text conversion.

Nevertheless, statistical sequence-to-sequence models have been developed to map the observation utterance $\mathbf{O}$ to word sequence $\mathbf{W}$. The majority of current systems take a generative approach to obtain the optimal classifier that maximises the posterior probability of the word sequence $P(\mathbf{W}|\mathbf{O})$.

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} P(\mathbf{W}|\mathbf{O}) \tag{2.1}$$

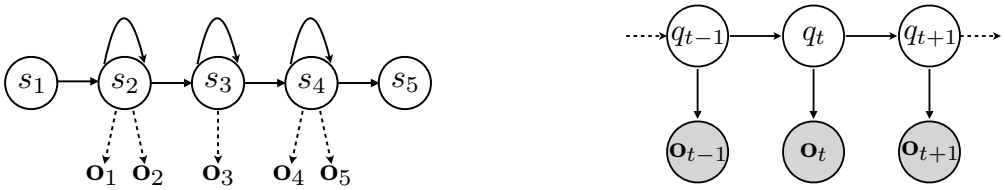
This form of the classifier is also referred to as a maximum a posteriori (MAP) classifier and can be further expanded to yield the equivalent decision rule

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} \frac{p(\mathbf{O}|\mathbf{W})P(\mathbf{W})}{p(\mathbf{O})} = \arg\max_{\mathbf{W}} \underbrace{p(\mathbf{O}|\mathbf{W})}_{\text{AM}} \underbrace{P(\mathbf{W})}_{\text{LM}} \tag{2.2}$$

The denominator $p(\mathbf{O})$ is omitted as it is constant for any given utterance, which is irrelevant for decision making. $p(\mathbf{O}|\mathbf{W})$ is the likelihood of an observation given a word sequence, which is computed by the acoustic model (AM). $p(\mathbf{W})$ is the prior distribution of a word sequence represented by a language model (LM). These two distributions on the right-hand side of Equation 2.2 are generally unknown and estimated from the training data described in the following sections.

2.1 Acoustic Modelling

The acoustic model is trained to model the conditional likelihood $p(\mathbf{O}|\mathbf{W})$ with a large corpus of utterances and the associated transcriptions. In ASR, hidden Markov models (HMMs) are widely used as illustrated in Fig. 2.1.



(a) Finite state machine representation. The initial and final states are non-emitting states.

(b) Graphical model representation. Shaded nodes are observed.

Fig. 2.1 The hidden Markov model.

In HMMs, $\boldsymbol{\theta}$ is a collection of state transition probabilities a_{s_i,s_j} and emission probabilities $b_{s_i}(\mathbf{o}_t)$. HMMs follow first-order Markov assumption (Equation 2.3) and state conditional independence assumption (Equation 2.4).

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

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

Therefore, the observation likelihood can be obtained by marginalising the likelihood over all possible hidden sequences

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

In practice, HMMs are typically trained on sub-word units, *e.g.* phones and context-dependent phones, due to the large size of vocabulary and insufficiency of data to build the acoustic model at word-level. A composite HMM for the entire word sequence can be formed by concatenating the phone-level HMMs. More advanced discriminative training criteria have been developed over the past decade to yield the state-of-the-art performance.

2.2 Language Modelling

The language model describes the prior probability of a particular word sequence $\mathbf{W}$ of length T.

$$P(\mathbf{W}) = P(w_1, w_2, \ldots, w_T) = P(w_1)P(w_2|w_1)\ldots P(w_T|w_{T-1}, \ldots, w_1) \tag{2.6}$$

As shown in Equation 2.6, the size of the distribution table grows exponentially w.r.t. the length of the sequence and become nearly impossible to have a good estimate even for short sequences due to the size of the vocabulary. With a limited amount of training data, conditional independence must be imposed for $p(\mathbf{W})$ to be computationally feasible.

The simplest form of a language model is the uni-gram model, where the probability of a word in a sequence is independent of its context and is estimated by its frequency count in the training set. However, the order of words in a sequence becomes irrelevant due to the strong independence assumption. By incorporating history information of a word, n-gram LM could be obtained by considering the conditional distribution of a word given previous n words, *i.e.*

$$P(w_i|w_{i-1}, \ldots, w_1) \approx P(w_i|w_{i-1}, \ldots, w_{i-n}) \tag{2.7}$$

There is a trade-off between the computational complexity and the quality of estimation w.r.t. the order of the n-gram model. However, as the length of context becomes the longer, the statistics become less accurate for a limited amount of data since the number of examples for a particular sequence is exponentially less frequent. Advanced techniques including statistical smoothing, interpolation and back-off are used to improve the generalisation of the LM.

Recently, neural language models have gained significant popularity as it circumvents the curse of dimensionality by encoding each word as a feature vector and expressing the joint probability of the sequence in terms of these feature vectors via a neural network. The neural network is trained as a probabilistic classifier to predict the probability distribution $P(w_i|\text{context})$, where the context can be the history and/or future words. The word feature vectors are also known as word embeddings that map from a sparse space with one dimension per word to a continuous vector space in much lower dimensions.

2.3 Decoding

By having the acoustic and language models, the remaining task of ASR is to maximise the word sequence posterior probability w.r.t. all possible word sequences according to the MAP rule, as indicated in Equation 2.2. Because of the large vocabulary size and potentially lengthy sequences, the naive approach of enumerating all possible word sequences is computationally infeasible. Viterbi approximation is used to retrieve the one-best sequence in the search space.

In the acoustic model, the probability densities of the likelihood are assumed to be independent across different HMMs, resulting in the acoustic model likelihoods being underestimated. To compensate for this effect on MAP decision rule in Equation 2.2, the decoding function includes an exponentially scaled language model probability and an insertion penalty.

$$f(\mathbf{W}) = p(\mathbf{O}, \hat{\mathbf{q}}|\mathbf{W})P(\mathbf{W})^{\gamma}\rho^{||\mathbf{W}||} \tag{2.8}$$

where $\hat{\mathbf{q}}$ is the best state sequence in the HMM search network, γ is the grammar scale factor (GSF) and ρ is the insertion penalty, the values of which are task-specific and could be fine-tuned to yield optimal decoding results.

2.3.1 Lattices

Lattices are compact representations of all the hypotheses made by the speech recogniser. One lattice is most likely to contain all the competing hypotheses of an observed utterance at the word level [20]. Topologically, a lattice is a directed acyclic graph (DAG) $\mathcal{G}$ with nodes/vertices $\mathcal{N}$ and edges/arcs $\mathcal{E}$. Each arc e corresponds to a word hypothesis proposed by the ASR system and contains information including the acoustic model score $p(\mathbf{O}|w)$, the language model score $p(w|\text{context})$, the pronunciation variant, and the identity of the source (or parent) node and the destination (or child) node. Each node N encodes the timestamp information and it may be connected to an arbitrary number of incoming arcs and outgoing arcs, which imposes the structure of the lattice. A path in the lattice represents a hypothesised word sequence and all paths are uni-directional with no loops. Visualisation of a word lattice from a short utterance is shown in Fig. 2.2.

Lattice Pruning

A lattice could be potentially very large, with thousands of nodes and edges. In order to reduce the size of the lattice and restrict the search space, lattice pruning procedures are normally used. Two major techniques for lattice pruning are briefly discussed as follows.

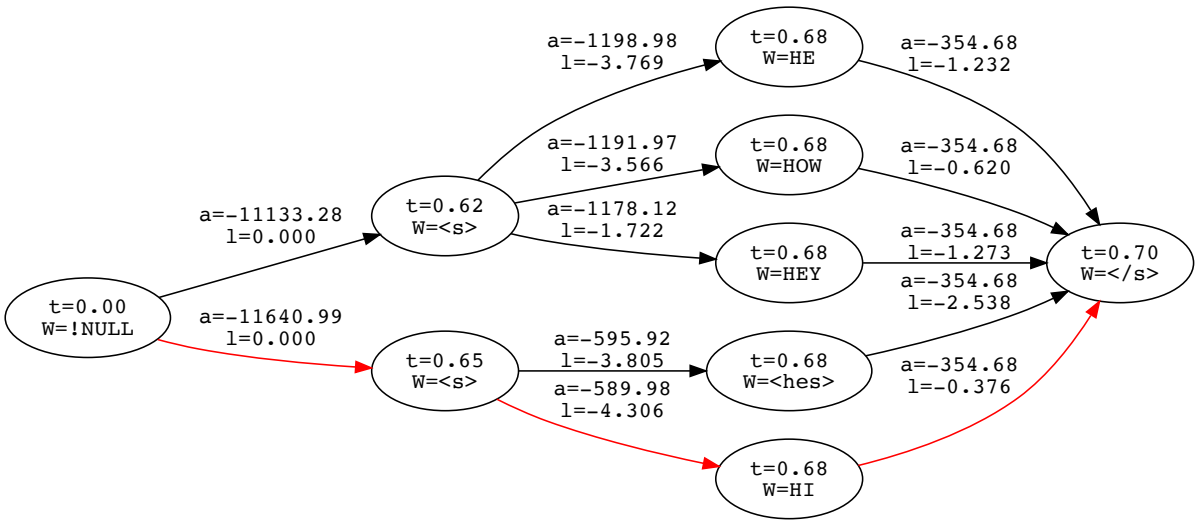


Fig. 2.2 An example of a word lattices. The path shown in red is the one-best path produced by Viterbi search.

- Beam pruning. At each time instant, the decoder discards all arcs whose likelihoods fall below a certain threshold or limit the number of arcs to a certain number, *i.e.* beam width, according to their likelihoods.
- Unique arcs per second (UAPS) pruning. At each time instant, there might be a large number of arcs with identical words but slightly different start and/or end timestamps with similar likelihood. This pruning technique merges similar arcs with similar timestamp information and tries to maintain a limited number of unique arcs within a certain time interval.

At each particular time instant, arcs in the beam-pruned lattices may have similar likelihoods whereas arcs in the USPS-pruned lattices may have more distinct likelihoods over a wide range of values. In other words, if a cross section is cut through the lattice and the likelihoods across arcs are viewed as a distribution, the UAPS-pruned lattices generally have a sharper distribution than its counterpart.

2.3.2 Confusion Networks

Confusion networks are an alternative dense representation of the most likely hypotheses in lattices. The confusion network is also known as “sausage” because of its constraint that all paths in a confusion network pass through all nodes. Topologically, confusion networks are a DAG-like structure where all the outgoing edges of a node are the incoming edges for the next node. The nodes segment the network at different timestamps and the set of arcs in between

any pair of adjacent nodes forms a confusion set. The arcs in a confusion network correspond to words, some of which are null words inserted to comply with the special constraint.

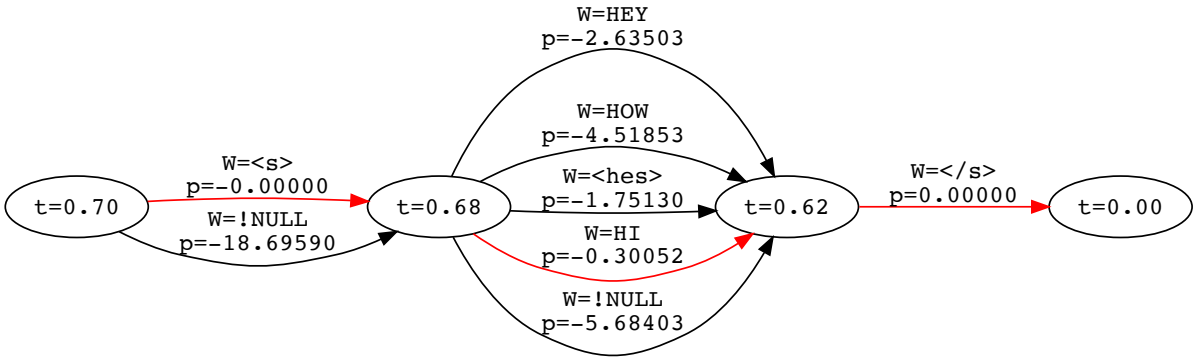


Fig. 2.3 An example of a confusion network. The path shown in red is the one-best path.

In terms of the information stored in confusion networks, they discard a large number of arcs with low likelihoods in lattices. However, confusion networks also create some new sequences which are not presented in lattices by imposing the constraint.

Chapter 3

Confidence Scores in ASR

Confidence scores provide an indication of the reliability of the transcription given by the recognition system. This piece of information is of paramount importance for many ASR-related systems. Out-of-vocabulary detection and keyword spotting applications rely heavily on the measure of confidence [34, 37]. Confidence scores can also provide substantial information and insights for both in-system applications such as ASR system combination [5], unsupervised adaptation of acoustic models [1], and active learning [16], and downstream applications such as speaking assessment [36], dialogue systems [8] and machine translation [2].

The task of obtaining a good estimate of the recogniser's confidence is challenging and decades of research work has been devoted to improving this measure [11]. The core of all existing approaches is to explore methods to effectively extract, represent and process useful features from the recogniser to produce confidence scores. The most related feature to confidence scores in the ASR system is the word posterior probability [4], which is firstly described in this chapter. The posterior probabilities are used as a baseline, and other useful features are also considered. Appropriate models are then built to use these features to yield a better estimation of confidence.

3.1 Posterior Probabilities

By taking the advantage of the statistical nature of ASR systems, the posterior probabilities of the hypotheses could be considered to be a reasonable reflection of the true accuracy of the output transcription. As discussed in Chapter 2, the MAP decision rule chooses the sequence with the highest posterior probability. Intuitively, multiple competing hypotheses with similar posterior probabilities are a strong indication that the recogniser is confused and uncertain about the result. Conversely, if one hypothesis has much higher posterior probability than all others, it is reasonable to conclude that the system is confident about this particular hypothesis.

As described in Section 2.3, the intermediate output of the speech recogniser could be represented efficiently using compact structures such as lattices and confusion networks. Both of these structures contain rich information related to the decoding process. Since each arc in these graphs corresponds to a word hypothesis, arc posterior probability could be derived and could be directly used as the confidence score. Approaches to calculating such arc posteriors are described as follows.

3.1.1 Lattice Arc Posteriors

For each arc in the lattice, it is associated with an acoustic model score and a language model score defined in § 2.3.1. For any path in the lattice $\mathbf{q}$, the joint lattice path probability can be expressed as

$$p(\mathbf{q}, \mathbf{O}) = \underbrace{p(\mathbf{O}|\mathbf{q})^{\frac{1}{\gamma}}}_{\text{AM}} \underbrace{P(\mathbf{W})}_{\text{LM}} \tag{3.1}$$

where the language scale factor is used to scale down the acoustic model score rather than scaling up the language model score. It is important because the acoustic model severely underestimates the emission probabilities due to the independence assumption. The lattice arc posterior probability of an arc e can then be defined as

$$p(e|\mathbf{O}) = \frac{\sum_{\mathbf{q} \in \mathcal{Q}_e} p(\mathbf{q}, \mathbf{O})}{p(\mathbf{O})} \tag{3.2}$$

where $\mathcal{Q}_e$ is a set of lattice paths that passes through the arc e and $p(\mathbf{O})$ is estimated by summing over all paths in the lattice. For the summation to be computed efficiently, the forward-backward algorithm is used. Forward probability α and backward probability β (in the log domain) are stored for each node and edge in the lattice. Generally, to compute the arc posterior probability for an arc e in an arbitrary lattice (Fig. 3.1):

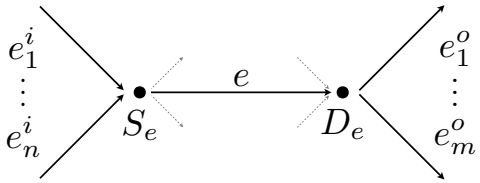


Fig. 3.1 An arc in a lattice. The source node S_e has n incoming arcs and the destination node D_e has m outgoing arcs.

- Initialisation

$$\alpha_{\text{root node}} = \texttt{LOGZERO} \tag{3.3}$$

$$\beta_{\text{end node}} = \texttt{LOGZERO} \tag{3.4}$$

- Forward (joint) probability

$$\alpha_{S_e} = \log\left(\sum_{j=1}^{n} \exp(\alpha_{e_j^i})\right) \tag{3.5}$$

$$\alpha_e = \alpha_{S_e} + \frac{1}{\gamma}\log(\mathbf{AM}_e) + \log(\mathbf{LM}_e) \tag{3.6}$$

  where S_e denotes the source node of arc e and $\{e_1^i, \ldots, e_n^i\}$ is the set of incoming arcs of the source node of arc e.

- Backward (conditional) probability

$$\beta_{D_e} = \log\left(\sum_{k=1}^{m} \exp\left(\alpha_{e_k^o} + \frac{1}{\gamma}\log(\mathbf{AM}_{e_k^o}) + \log(\mathbf{AM}_{e_k^o})\right)\right) \tag{3.7}$$

$$\beta_e = \beta_{D_e} \tag{3.8}$$

  where D_e denotes the destination node of arc e and $\{e_1^o, \ldots, e_m^o\}$ is the set of outgoing arcs of the destination node of arc e.

- Arc posterior probability

$$\log p(e|\mathbf{O}) = \alpha_e + \beta_e - \log Z \tag{3.9}$$

  where $Z = \sum_{e' \in \mathcal{E}} \exp(\alpha_{e'} + \beta_{e'})$ is the normalisation constant which ensures the sum-to-one constraint for a valid probability distribution. The normalisation term $p(\mathbf{O})$ is obtained at the end of forward passing $\alpha_{\text{end node}}$ or equivalently, the end of backward passing $\beta_{\text{root node}}$.

Note that for numerical stability when adding in the log domain, the following identity is used:

$$\log(x+y) = \max(x,y) + \log\left(1 + \exp\left(\min(x,y) - \max(x,y)\right)\right) \tag{3.10}$$

3.1.2 Confusion Network Arc Posteriors

An algorithm for generating confusion networks from lattices is briefly described below [17].

1. The lattice arc posterior probabilities are computed as in Equation 3.2.

2. Time alignment. The posterior probabilities for each word in the lattice with the given start time t_s and the end time t_e are calculated using $P(W|t_s, t_e, \mathbf{O})$. Each word hypothesis w_i in the word sequence represents a cluster.

3. Intra-word clustering. Clusters corresponding to the same word which overlap in time are merged. The word posterior probabilities become available.

4. Inter-word clustering. Clusters corresponding to different words that are considered as belonging to the same confusion set are merged. The merge is based on the time overlap between words and a phonetic similarity score weighted by the posterior probability. The required structure is obtained and the competing word hypotheses can be identified.

5. Pruning. Null words are added to the structure and their associated posterior probabilities are the remaining of the probability mass in each confusion set, *i.e.* all posteriors in a confusion set sum to 1.

3.2 Standard Approaches

Apart from using arc posteriors directly from either lattices or confusion networks, the estimation could be improved by considering other related information generated during the decoding process. With appropriate models, effective features can be captured to make more accurate predictions on confidence scores. In this section, features considered by previous work are summarised and effective models are described.

3.2.1 Features

A number of direct and derived features from the recognition pass have been proposed in the literature. Ideally, all features used to estimate the confidence scores are complementary with each one providing some evidence about the difference between the correctly hypothesised words and their counterparts. If these features are extracted and combined effectively, the confidence scores are expected to be highly reliable. Some features available during the decoding process which could be used as a part of the estimator are categorised as follows.

- Lattice-based features:

  – hypothesis density [14] – a measure based on the number of alternative arcs that span the given time interval. The higher the hypothesis density, the lower the confidence

score is likely to be. However, as mentioned in § 2.3.1, pruning is normally conducted on lattices. The pruning method directly influences the effectiveness of this feature.

  - word trellis stability [23] – a measure based on the observation that a word is more likely to be correct if it appears in the majority of the most probable hypotheses within a particular time interval, as incorrect words are more sensitive to variations of the grammar scale factor.

- Acoustic model-based features:
  - normalised acoustic likelihood [21] – the acoustic model score divided by the number of frames or the number of phones, which allows the acoustic model scores to be comparable across different word hypotheses.
  - acoustic stability [39] – a measure based on the number of times a given word occurs in the same aligned position by varying the GSF.
  - sub-word level acoustic scores [33]
- Language model-based features:
  - language model score – as stated in Section 2.1.
  - language model back-off behaviour [29] – a measure that assumes the language model becomes less confident when it has to back-off to a lower order of n-gram to have a more reliable estimation.
- Duration-based features [29]: HMM state duration, phone duration, or word duration based on the premise that shorter duration typically corresponds to regions of poor performance of the acoustic model.
- Utterance-based features [8]: measures including word position, length of utterance, and lexical identity of hypothesised word sequences, with similar rationale as language model-based features.

3.2.2 Models

Despite a large number of available features, most of them are more or less correlated to each other [13]. Therefore, powerful models are expected to project these features into an informative space for confidence score estimation. In the past, a wide selection of models has been applied to combine multiple features and to boost their discriminative power. Some of the

statistical classifiers used for feature combination include decision trees, linear functions or models, Gaussian classifiers, neural networks, boosting, naive Bayes, support vector machines, and maximum entropy models. As one of the most recent works, conditional random fields (CRFs) are briefly described below.

Conditional Random Field

CRFs are a probabilistic discriminative model that could combine multiple features to predict confidence scores of a sequence. A simple linear chain CRF is shown as an undirected graphical model in Fig. 3.2. The distribution modelled by CRFs is

$$P(y_1, \ldots, y_T | \mathbf{x}_1, \ldots, \mathbf{x}_T) \propto \exp\left(\sum_{t=1}^{T}\left(\sum_{d=1}\lambda_d f_d(y_t, y_{t-1}) + \sum_{d=1}\mu_d g_d(y_t, \mathbf{x}_t)\right)\right) \quad (3.11)$$

where $f(\cdot,\cdot)$ and $g(\cdot,\cdot)$ are the feature functions. The parameters are estimated by maximum likelihood learning.

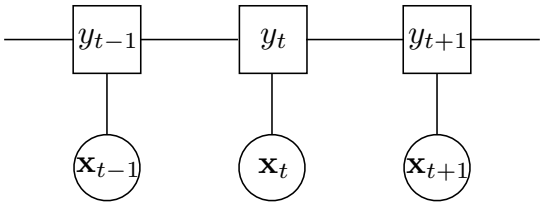


Fig. 3.2 The linear chain CRF.

Some properties of CRFs are very useful for sequence modelling such as estimation of confidence scores [26]. The input feature $\mathbf{x}_t$ can be a combination of arbitrary features from the input sequence. The label sequence is conditioned on the whole observation sequence. Therefore, long-term dependencies on both sides of a word in a sequence are modelled. Other variants of CRFs with greater complexity, including higher-order CRFs, hidden state CRFs and hierarchical CRFs, could also be used.

3.2.3 Comparison of Approaches

In the context of confidence score estimation, the following aspects are discussed for the above approaches [25]:

- Sources of information. By solely using the word posteriors in lattices or confusion networks as confidence score, the information is already available in the ASR system. Some other derived features from the decoding process require either additional heuristics

or other systems, *e.g.* stability-related features. It is desirable to have an approach that takes the raw information from the recogniser without expert knowledge for confidence score estimation.

- Complexities. Using word posteriors directly requires minimal effort. However, linear and non-linear models are studied to yield better confidence scores by combining multiple input features. The complexity varies from simple models such as the decision tree, to some simple probabilistic models such as naive Bayes, to complex models including CRFs.

- Score granularity. Usually, the confidence scores are defined at the word level. In some applications, sub-word level or utterance level confidence scores may be required. Although using arc posteriors could extend to the sub-word level, it is not applicable at the utterance level. Employing a classification-based model is a more appropriate approach to take feature vectors and predict confidence scores at any level of granularity.

- Sequentiality. Many models mentioned above can only classify a defined input feature vector, *e.g.* Gaussian classifiers and support vector machines. However, these approaches largely ignore the information encoded in the data structure. Models that could utilise information from the past or the future are more suitable.

- In-system applications. A model that operates directly on lattices is preferred. Because the confidence can be readily integrated into the system, such as lattice rescoring [4] and unsupervised acoustic model adaptation [1]. Most classifiers mentioned are difficult for such applications.

In summary, an ideal model for confidence score estimation would be a classification-based model that could operate on lattices directly and could incorporate any raw information contained in lattices. LATTICERNN is introduced in the next chapter as a solution to the problem.

Chapter 4

Deep Learning Models for Confidence Scores

In order to have more reliable confidence scores, word posteriors and multiple predictor features from the decoding process should be optimally combined. Since all features mentioned in § 3.2.1 are more or less correlated, and hand-crafted features and feature functions are subject to strong assumptions, the design of a general statistical framework that could extract useful information for confidence estimation from an arbitrary number of highly dependent features is desirable. § 3.2.2 introduces CRF as a sequential classification model for confidence scores, which has been proved to be effective in combining multiple features with different types, *i.e.* discrete and continuous values without making independence assumptions between information sources. Instead of statistical models, artificial neural networks (ANNs) have been predominant in recent research works on classification tasks due to their capability to model non-linear processes. Recurrent neural networks (RNNs) is a class of ANN architectures, which has been widely used for sequential data [12].

All previous models are applied to linear sequences where features of each word are transformed by the model to predict the corresponding word confidence. However, this process neglects a large amount of information from the decoder of ASR systems, namely the competing hypotheses, by only using the information from the one-best sequences. In this chapter, a structured deep learning model, LATTICERNN [15, 27], is introduced as a flexible framework that could incorporate an arbitrary number of features with different types, and can be applied not only on linear chains but more generally on any DAG-like structure. LATTICERNN can also operate on the sub-word level or the utterance level, and can be integrated for in-system applications.

4.1 Recurrent Neural Networks

RNNs model sequential data by using internal hidden states which can be considered as a representation of history inputs. For a generic RNN model shown in Fig. 4.1, each unit applies an element-wise non-linearity to the affine transformation of the previous hidden state and the current input (Equation 4.1). The output of each time step is a function of the current hidden state (Equation 4.2).

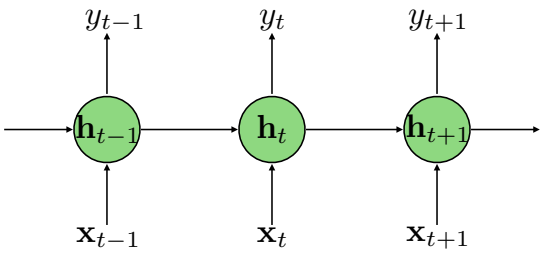


Fig. 4.1 The unrolled RNN model.

$$\mathbf{h}_t = \phi(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{h}_{t-1} + \mathbf{b}) \tag{4.1}$$

$$y_t = \mathcal{F}(\mathbf{h}_t) \tag{4.2}$$

where $\mathbf{U}, \mathbf{W}, \mathbf{b}$ are input weights, recurrent weights and bias shared across different time steps, $\phi(\cdot)$ is the activation function, and $\mathcal{F}(\cdot)$ is any valid function that maps from the hidden state to the output space.

By unrolling the RNN in time, the model can be viewed as a very deep fully connected neural network with weights tied across all layers. Similar to other building blocks for deep learning, multiple recurrent layers can be stacked together to form a deep architecture. Instead of using standard error backpropagation to train feed-forward networks, backpropagation through time (BPTT) [31] is used.

4.1.1 LSTM

In practice, when the input sequence grows longer, the simple RNN structure suffers from vanishing gradients problem when the network is trained using gradient descent algorithms, *i.e.* long-term information is lost. To address this issue, long short-term memory (LSTM) model [10] uses multiple gating functions and an additional internal recurrence (the cell state $\mathbf{c}$), where the gradient can flow for longer durations. LSTM is a more complex model inherited from Fig. 4.1, where each recurrent unit is replaced by the structure shown in Fig. 4.2. The weights of the internal recurrence is controlled by a forget gate unit $\mathbf{f}$. The external input gate $\mathbf{i}$

is computed in a similar fashion as the forget gate, but with its own parameters. The hidden state $\mathbf{h}$ is manipulated by the output gate $\mathbf{o}$ on the updated cell state. All three gating functions produce values between 0 and 1 via the sigmoid function $\sigma(\cdot)$.

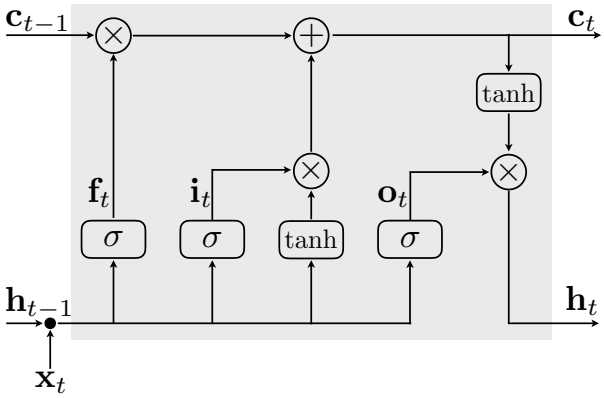


Fig. 4.2 The LSTM cell structure.

$$\mathbf{f}_t = \sigma\Big(\mathbf{U}^f\mathbf{x}_t + \mathbf{W}^f\mathbf{h}_{t-1} + \mathbf{b}^f\Big) \tag{4.3}$$

$$\mathbf{i}_t = \sigma\Big(\mathbf{U}^i\mathbf{x}_t + \mathbf{W}^i\mathbf{h}_{t-1} + \mathbf{b}^i\Big) \tag{4.4}$$

$$\mathbf{o}_t = \sigma\Big(\mathbf{U}^o\mathbf{x}_t + \mathbf{W}^o\mathbf{h}_{t-1} + \mathbf{b}^o\Big) \tag{4.5}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh\Big(\mathbf{U}^c\mathbf{x}_t + \mathbf{W}^c\mathbf{h}_{t-1} + \mathbf{b}^c\Big) \tag{4.6}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh\big(\mathbf{c}_t\big) \tag{4.7}$$

where $\odot$ indicates element-wise product.

4.1.2 Bidirectional LSTM

RNN or LSTM has a linear chain structure where the recurrence only depends on the current input and the history. However, when a complete sequence is available, incorporating information from the future is useful because the prediction could be more accurate when the network observes both the past and future contexts. A bidirectional recurrent neural network [24] achieves this by concatenating the hidden states of two RNN layers, with one processing the sequence forwards and the other one backwards. In the context of LSTM, the bidirectional form can be expressed as follows.

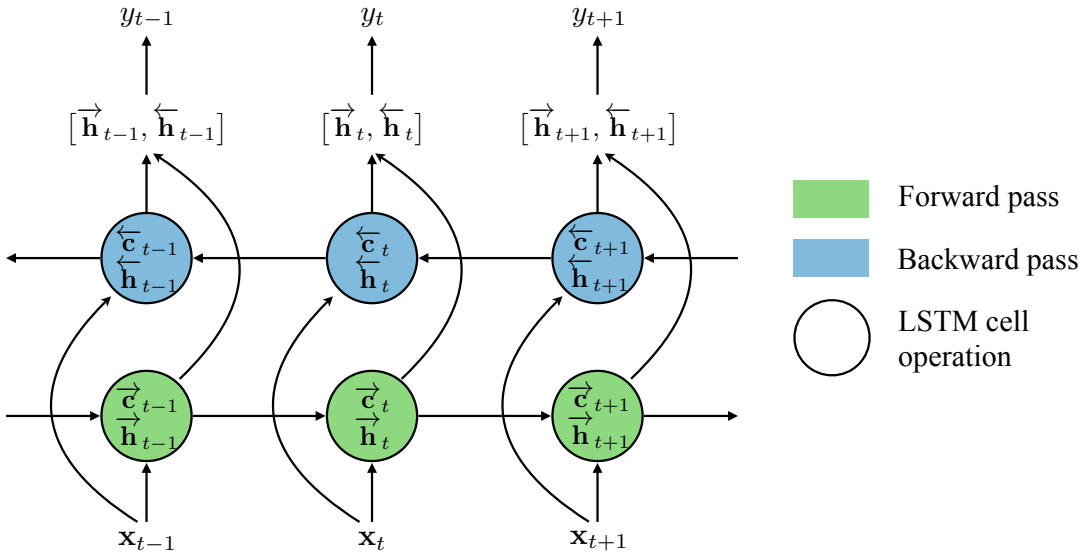


Fig. 4.3 The unrolled BLSTM model.

$$\overrightarrow{\mathbf{h}}_t = \text{LSTM}(\overrightarrow{\mathbf{h}}_{t-1}, \mathbf{x}_t, \overrightarrow{\mathbf{c}}_{t-1}) \tag{4.8}$$

$$\overleftarrow{\mathbf{h}}_t = \text{LSTM}(\overleftarrow{\mathbf{h}}_{t+1}, \mathbf{x}_t, \overleftarrow{\mathbf{c}}_{t+1}) \tag{4.9}$$

$$y_t = \mathcal{F}(\overrightarrow{\mathbf{h}}_t, \overleftarrow{\mathbf{h}}_t) \tag{4.10}$$

BLSTM is a suitable model for confidence score estimation on one-best hypotheses. Each word and its associated features in the one-best sequence can be combined as the input feature $\mathbf{x}$ to the network. Forward and backward passes allow the context in the one-best sequence to be fully captured.

4.2 LATTICERNN

Compared with lattices and confusion networks, one-best sequences contain the least amount of information about all other competing hypotheses produced by ASR systems. In order to incorporate the information missed by one-best sequences, previous models based on linear chains must be adapted to accommodate more general data structures, *e.g.* DAGs. To this end, LATTICERNN , a form of recursive recurrent neural networks, is employed. LATTICERNN traverses the graph in its topological order [15] and the detailed information flow is described in the following sections. The designed characteristics of LATTICERNN include

- the input can be any valid directed acyclic graphs of variable structures;
- the feature vector associated with each arc could contain an arbitrary number of features mentioned in § 3.2.1;

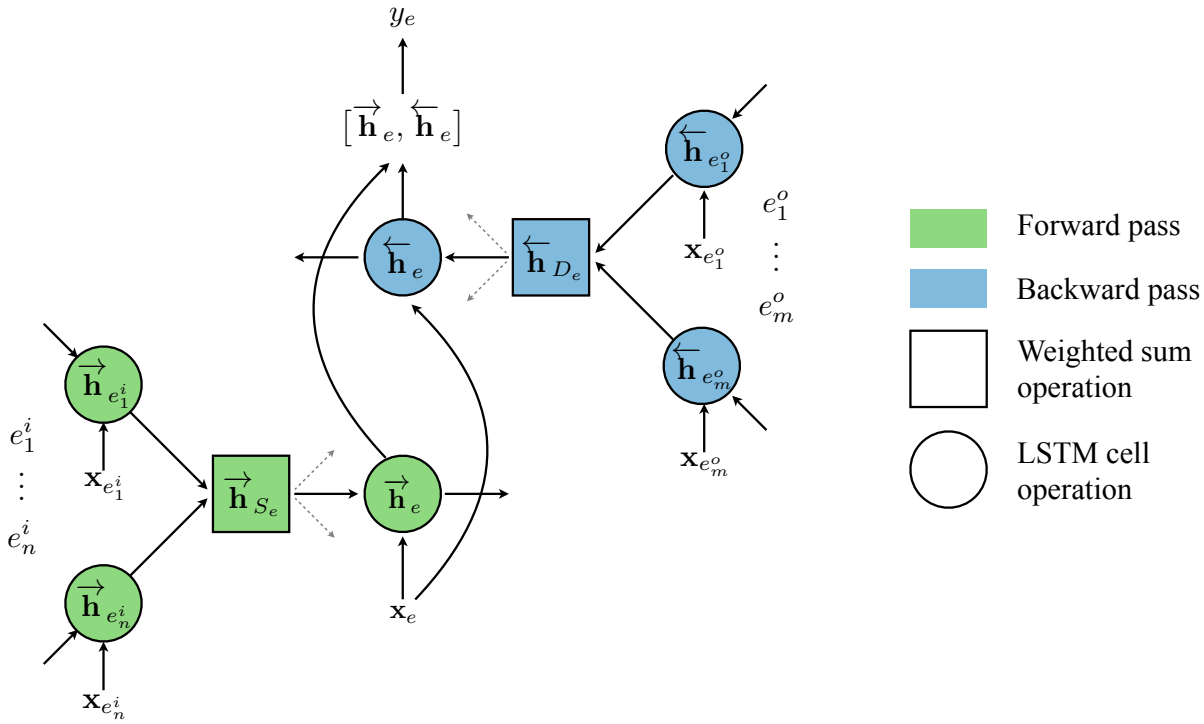


Fig. 4.4 The LATTICERNN model.

- each dimension of the feature vector is not subject to independence assumption;
- the model could be applied to different levels of granularity, *i.e.* the sub-word level, the word level and the utterance level;
- the model could capture both long term and short term dependencies using context in the structured input.

4.2.1 Model

The structure of the LATTICERNN is inspired by the forward-backward algorithm for computing arc posteriors in the lattice described in § 3.1.1 and the LSTM network for modelling sequential information. For each arc e in the lattices or confusion networks, multiple features could be concatenated into the feature vector $\mathbf{x}_e$.

LATTICERNN applies LSTM-like recurrence on the lattice structure in a topological order both forwards and backwards as shown in Fig. 4.4. Different from a linear chain, each node and each edge in the graph have their associated hidden states.

In a forward pass, the hidden state of an arc $\overrightarrow{\mathbf{h}}_e$ is the output of the recurrence with the input being the hidden state of its source node $\overrightarrow{\mathbf{h}}_{S_e}$ and the input feature vector $\mathbf{x}_e$. The hidden

state of the source node $\overrightarrow{\mathbf{h}}_{S_e}$ is a weighted sum of the hidden states of n incoming arcs $\overrightarrow{\mathbf{h}}_{e_j^i}$ for $j = 1, \ldots, n$.

$$\overrightarrow{\mathbf{h}}_{S_e} = \sum_{j=1}^{n} a_j \overrightarrow{\mathbf{h}}_{e_j^i} \tag{4.11}$$

$$\overrightarrow{\mathbf{h}}_{e} = \text{LSTM}(\overrightarrow{\mathbf{h}}_{S_e}, \mathbf{x}_e) \tag{4.12}$$

where a_j are the weights for the forward weighted sum operation where $\sum_j a_j = 1$.

Similarly in a backward pass, the hidden state of an arc $\overleftarrow{\mathbf{h}}_{e}$ is the output of the recurrent unit which takes the hidden state of its destination node $\overleftarrow{\mathbf{h}}_{D_e}$ and the input feature vector $\mathbf{x}_e$. The hidden state of the destination node $\overleftarrow{\mathbf{h}}_{D_e}$ is a weighted sum of the hidden states of m outgoing arcs $\overleftarrow{\mathbf{h}}_{e_k^o}$ for $k = 1, \ldots, m$.

$$\overleftarrow{\mathbf{h}}_{D_e} = \sum_{k=1}^{m} b_k \overleftarrow{\mathbf{h}}_{e_k^o} \tag{4.13}$$

$$\overleftarrow{\mathbf{h}}_{e} = \text{LSTM}(\overleftarrow{\mathbf{h}}_{D_e}, \mathbf{x}_e) \tag{4.14}$$

where b_k are the weights for the backward weighted sum operation where $\sum_k b_k = 1$.

After finishing the forward pass and the backward pass, the hidden representation of an arc is the concatenation of the forward hidden state $\overrightarrow{\mathbf{h}}_{e}$ and the backward hidden state $\overleftarrow{\mathbf{h}}_{e}$. The output prediction y_e is a function of this hidden representation.

$$y_e = \mathcal{F}(\overrightarrow{\mathbf{h}}_{e}, \overleftarrow{\mathbf{h}}_{e}) \tag{4.15}$$

Table 4.1 highlights the analogies and distinctions of LATTICERNN comparing to the forward-backward algorithm on lattices. It is worth noting that there is no normalisation term in LATTICERNN as the $p(\mathbf{O})$ term in forward-backward algorithm. In the forward-backward algorithm, since the probabilities are additive at merging point and multiplicative as the probabilities are pushed through the lattice, global normalisation is required to yield a valid probability distribution for arc posteriors. However, at each node in LATTICERNN where there are multiple incoming arcs, the hidden states are summed with weights that adds up to 1. The recurrent unit also ensures that the range of the hidden state to be consistent. By normalising at each merging point and keeping the range to be invariant during message propagation, the overall normalisation is no longer necessary. Unlike the posterior probabilities in the forward-backward algorithm, LATTICERNN encodes the messages as hidden states, which are not readily interpretable.

	forward-backward algorithm	LATTICERNN
input	AM & LM scores	feature vector
message content	path log-probability	hidden state
message propagation	probability accumulation	recurrent unit
message merging	log add	weighted average
message flow	forward & backward	bidirectional
normalisation	explicit	implicit
output	arc posteriors	hidden representation

Table 4.1 Comparison between forward-backward algorithm and LATTICERNN .

4.2.2 Arc Merging Functions

For the weighted sum at each node with multiple incoming arcs, the following functions that yields positive weights with sum-to-one constraint are used:

- Max function based on arc posteriors

$$a_j = \begin{cases} 1 & \text{if } j = \arg\max_{j'} p(e^i_{j'}|\mathbf{O}) \\ 0 & \text{otherwise} \end{cases} \qquad b_k = \begin{cases} 1 & \text{if } k = \arg\max_{k'} p(e^o_{k'}|\mathbf{O}) \\ 0 & \text{otherwise} \end{cases} \tag{4.16}$$

- Mean function

$$a_j = \frac{1}{n} \qquad b_k = \frac{1}{m} \tag{4.17}$$

- Normalised arc posteriors [15]

$$a_j = \frac{p(e^i_j|\mathbf{O})}{\sum_{j'} p(e^i_{j'}|\mathbf{O})} \qquad b_k = \frac{p(e^o_k|\mathbf{O})}{\sum_{k'} p(e^o_{k'}|\mathbf{O})} \tag{4.18}$$

The max function can effectively reduce the lattice to the one-best sequence as the max operation on arc posteriors follows the same procedure as the Viterbi algorithm. It will give no advantage of LATTICERNN over BLSTM models as it incorporates no extra information. The mean function assumes that the hidden state of the arc itself carries its importance, *i.e.* the magnitude of each dimension would be smaller if it believes this arc provides little evidence for confidence scores. However, this assumption is hard to be justified or to be imposed on the network. Using normalised arc posteriors as weights is a more sensible approach where the importance of each arc is assumed to be proportional to the arc posteriors. If the arc posterior is relatively high, then the speech recogniser believes the corresponding word is more

likely to be correct, hence high confidence. However, this correlation between significance and arc posteriors may not be linear. To dynamically allocate weights on merging arcs, attention mechanism, which is particularly successful in image captioning [35] and machine translation [28], could be used.

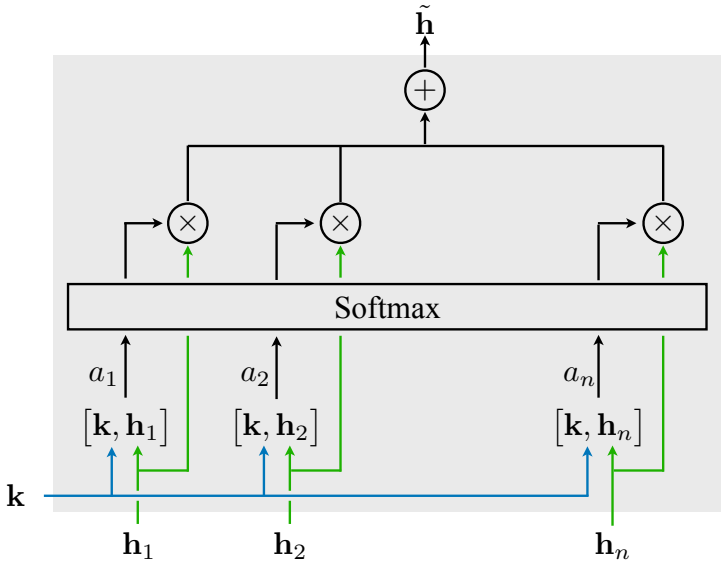


Fig. 4.5 The attention mechanism.

As shown in Fig. 4.5, the general attention mechanism takes all the inputs about to be combined, along with a common key vector $\mathbf{k}$. The concatenated vector of each input and the key vector yields a scalar via some non-linear transform. $a_1, \ldots, a_n$ are then passed through a softmax layer which normalises all the weights into a valid probability mass function. For LATTICERNN , the number of inputs varies as the number of incoming arcs are different at each node, but the key vector must be of fixed length. One of the appropriate key vectors for confidence scores is the first and second moments of the arc posterior probabilities of all the merging arcs. Therefore, the concatenated input for forward merging becomes

$$\Big[\underbrace{\mu_{e^i}, \sigma_{e^i}, p(e^i_j|\mathbf{O})}_{\mathbf{k}_j}, \underbrace{\overrightarrow{\mathbf{h}}_{e^i_j}}_{\mathbf{h}_j}\Big] \tag{4.19}$$

where μ_{e^i} and σ_{e^i} are the mean and standard deviation of the incoming arc posteriors. The attention mechanism for backward merging is similar.

4.2.3 Training

During the forward pass, nodes in a lattice are visited following the topological order. In Algorithm 1, `topological_sort()` gives the list of nodes following topological order on the input graph; `in_arcs()` and `out_arcs()` returns a set containing all the incoming arcs and outgoing arcs w.r.t. to the node of interest respectively; `arc_merge()` is a function described in § 4.2.2 that merges the hidden states of input arcs; and `LSTM()` is the LSTM recurrence function.

Algorithm 1 LATTICERNN forward pass

for node $n \in$ `topological_sort`$(\mathcal{G})$ **do**
  if `in_arcs`$(n) \neq \varnothing$ **then**
    $\overrightarrow{\mathbf{h}}_n =$ `arc_merge`(`in_arcs`(n))
  else
    $\overrightarrow{\mathbf{h}}_n = \mathbf{0}$
  end if

  for $e \in$ `out_arcs`(n) **do**
    $\overrightarrow{\mathbf{h}}_e =$ `LSTM`$(\overrightarrow{\mathbf{h}}_n, \mathbf{x}_e)$
  end for
end for

The backward pass is analogous, except the nodes are visited in the reversed topological order.

For error backpropagation, stochastic gradient descent (SGD) is used. Similar to training a BLSTM network by BPTT, the optimal parameters for LATTICERNN are found by backpropagation through structure (BPTS) [7] where the DAG topology of lattices becomes useful since a general recursion could be implemented on arbitrary input structures.

Chapter 5

Implementation

From data processing to network training, efficient data handling is of paramount importance. Computationally, the data structure of the lattice must be both compact for storage and flexible for bidirectional traversing. For supervised training, labels are needed for either the arcs on the one-best path or all the arcs in the lattice, which proved to be non-trivial. Since each training sample may have its individual structure, using batches or mini-batches is virtually impossible. Furthermore, the recursive nature of the network impedes the full advantage of GPU acceleration to be taken. This chapter reveals the implementation details that would allow LATTICERNN to work practically and efficiently.

Python is chosen to be the main language for prototyping as it is the most compatible language with the widely used and supported open-source deep learning libraries. PyTorch library[1] provides the core building blocks of deep learning modules for this project. One major advantage of PyTorch over other libraries, *e.g.* TensorFlow, Caffe, and CNTK, is that computation graphs are dynamic using reverse-mode auto-differentiation, which is well-suited for models with variable input structures such as LATTICERNN .

5.1 Data Preprocessing

Data preprocessing is used as an offline stage where the pre-computation of features and conversion of data structures are finished before being fed into the neural networks. Although it will take some disk space to store the preprocessed data, it significantly saves computational overhead for data processing upon loading and some global features could be obtained. Since there are no natural training labels available for words in lattices, alignment with the ASR reference is required.

[1] http://pytorch.org/

5.1.1 Data Structure

The data structure is compatible with both lattices and confusion networks. Different from lattices, confusion networks allows more than one arc between the adjacent pair of nodes. In this project, conversion tools are implemented to convert from compressed HTK-style lattice `.lat.gz` [38] and confusion network `.scf.gz` to a consistent and compressed NumPy format `.npz`. In the converted format, the following attributes are available:

- `topo_order` – a list of node indices that follows a topological order;
- `child_2_parent` – a dictionary that maps from a node index to a dictionary, whose key is the index of the parent node and the value is the index of the connecting edge for lattices or a list indices of the connecting edges for confusion network. This is used for the forward recursion;
- `parent_2_child` – a dictionary that maps from a node index to a dictionary, whose key is the index of the child node and the value is the index of the connecting edge for lattices or a list indices of the connecting edges for confusion network. This is used for the backward recursion;
- `edge_data` – a matrix containing all relevant information from the source file indexed by the edge index. For an arc in a lattice, the information includes the word, the start time and the end time, LM and AM scores. For an arc in a confusion network, the arc posterior probability, the start and the end time are available;
- `ignore` – a list of edge indices whose corresponding word is one of the following `<s>`, `</s>`, `!NULL`, `<hes>`, which are due to be skipped during training of the network.

5.1.2 Word Embeddings

To represent the word identity, using one-hot encoding would be inefficient when the vocabulary size gets large. Alternatively, word embeddings is a low-dimensional continuous space representation mapped from one-hot encoded words [19]. In this project, fastText model [3] is adopted to train the word embeddings in an unsupervised fashion. Building upon the skipgram model [18], this model represents each word as a bag of character n-grams. A vector representation is associated with each character n-gram and words being represented as the sum of these representations. FastText model allows sub-word information to be captured and consequently, word embeddings for out-of-vocabulary words could be computed [3]. It has been shown that the fastText model produces word vectors that achieve state-of-the-art performance on word

similarity and analogy tasks. Operating on the sub-word level, the word embeddings by fastText might contain implicit information such as the length of the word by adding character n-gram representations, and pronunciation similarity due to the close relation between sub-word strings and phonetics. FastText embeddings could be correlated to the acoustic model and could be complementary to the word-level language model used for recognition.

5.1.3 Data Normalisation

One advantage of LATTICERNN for confidence score estimation is the use of an arbitrary number of features with different dynamic ranges. Data whitening is employed by dividing each dimension of the data by its standard deviation after the mean has been removed. For numerical stability when computing the sum of squares, a two-pass algorithm is normally used. The first step is to compute the sample mean, and then the variance is computed as the sum of the squares of the differences from the mean. However, for a very large dataset, two passes might be expensive. An online algorithm is used to estimate the statistics in a numerically stable manner [30]. The recurrence relation is shown as follows.

$$\bar{x}_n = \bar{x}_{n-1} + \frac{x_n - \bar{x}_{n-1}}{n} \tag{5.1}$$

$$M_n = \sum_{i=1}^{n} (x - \bar{x}_{n-1})^2 = M_{n-1} + (x_n - \bar{x}_{n-1})(x_n - \bar{x}_n) \tag{5.2}$$

$$s_n^2 = \frac{M_n}{n-1} \tag{5.3}$$

$$\sigma_n^2 = \frac{M_n}{n} \tag{5.4}$$

where $\bar{x}_n$ denotes the sample mean, s_n^2 is the sample variance, and σ_n^2 is the population variance.

5.1.4 Arc Tagging

Since LATTICERNN is a supervised training task for classification, words associated with each arc in lattices and confusion networks need to be labelled to compute the loss function. There are two approaches for arc tagging:

- Partial tagging on one-best sequences using Levenshtein distance. For any lattice or confusion network, Viterbi algorithm could be employed to find the indices of arcs along the one-best path. Having the one best sequences and training transcriptions for the ASR system, the Levenshtein distance between two strings a, b could be computed as $\texttt{lev}_{a,b}(|a|, |b|)$, where $\texttt{lev}_{a,b}(i, j)$ is defined as

if $\min(i, j) = 0$, then

$$\texttt{lev}_{a,b}(i, j) = \max\{i, j\}$$

else

$$\texttt{lev}_{a,b}(i, j) = \min \begin{cases} \texttt{lev}_{a,b}(i-1, j) + 1 & \text{(deletion)} \\ \texttt{lev}_{a,b}(i, j-1) + 1 & \text{(insertion)} \\ \texttt{lev}_{a,b}(i-1, j-1) + 1_{(a_i \neq b_j)} & \text{(substitution if } a_i \neq b_j) \end{cases}$$

In the one-best sequences from lattices or confusion networks, all insertions and substitutions are tagged as 0 (incorrect) or 1 (correct) otherwise since no deletions will appear in one-best sequences.

- Full tagging on all arcs using timestamp information. Since each arc is associated with a start frame t_s and an end frame t_e, and the reference word-level time alignment (r_s, r_e) for the transcription is available, it is reasonable to classify the arc if the words are the same and there is some overlap between these two time intervals. The degree of overlapping is defined as

$$\eta = \max\left\{0, \frac{|\min\{r_e, t_e\}| - |\max\{r_s, t_s\}|}{|\max\{r_e, t_e\}| - |\min\{r_s, t_s\}|}\right\} \tag{5.5}$$

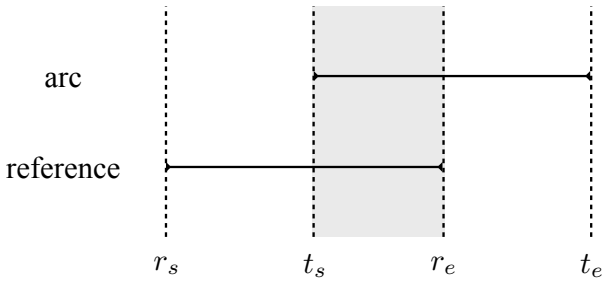


Fig. 5.1 An example of overlapping of two time intervals.

A threshold η_{th} between 0 and 1 could be set so that the arc is tagged as 1 (correct) if $\eta \geq \eta_{\text{th}}$ and 0 otherwise.

5.2 Training Procedure

5.2.1 Training Criterion

As in many other classification tasks using neural networks, binary cross entropy is adopted as the loss function for LATTICERNN

$$\mathcal{L} = \sum_{e \in \mathcal{E}} -y_e^* \log y_e - (1 - y_e^*) \log(1 - y_e) \tag{5.6}$$

where y_e^* is obtained from arc tagging in § 5.1.4. The error signals are only accumulated on the one-best path for partial tagging, but can also be on all arcs for full tagging.

5.2.2 Parallelism

Generally, the neural network is trained using SGD algorithms and the entire dataset is seen by the network over and over again until the loss converges. However, there are different granularities about how much data is provided to the network before parameters being updated. One end of the spectrum is called online learning, where the data samples are observed sequentially and the gradients based on the single observation are computed to update the parameters. The other extreme is called batch training where the gradient based on the entire dataset is computed before each update. The former method provides a poor estimate of the true gradient and results in noisy updates. In contrast, the latter approach is relatively expensive computationally. As a compromise, mini-batch updates are generally used. The bonus of using mini-batches is to exploit the parallel computing power of GPUs. Hundreds of samples are passed to the network and the gradients can be computed concurrently in matrix form.

However, in the case of LATTICERNN , each sample has its unique structure in terms of the forward/backward pass and the backpropagation process. Unlike the fully connected networks or the convolutional neural networks whose input features share the same dimensionality, lattices or confusion networks do not.

For neural network-based sequence models such as LSTM, input sequences of variable lengths could be allowed by padding the all sequences in one mini-batch as the same length as the longest one. It is true that the same idea could be applied to LATTICERNN such that the lattices or confusion networks in a mini-batch share an identical structure. However, the computational cost will rise dramatically. For lattices, the padding algorithm must ensure that all samples have the same depth (length of the longest sequence) and the same breath with identical connections at each level of depth. In the worst case, the padding could introduce $|\mathcal{N}_{\max}|^2$ number of new arcs where $\mathcal{N}_{\max}$ is the maximum number of nodes in a lattice within

the mini-batch. Although the computational cost for padded mini-batches is upper bounded by the computational cost of the longest sequence for RNNs, the overall computational cost may far exceed the individual cost of processing the most complex lattice.

To avoid complexity of padding, the online training procedure is followed. However, online training takes too long for the model to be used practically when the dataset is large. The speed-up offered by GPU concurrency is not significant because the data is not batched and the model itself is recursive. Parallelising SGD on multiple CPU cores in a lock-free manner would greatly enhance the throughput approximately by the number of cores available. This is called the `Hogwild!` approach [22]. As far as the modern high-performance computing infrastructure is concerned, occupying a few dozen CPU cores at a time is much less expensive than taking up a handful of GPU cores.

`Hogwild!` training circumvents the issue of memory locking and synchronisation across different cores that would diminish the proposition of parallelism. However, by assuming that the optimisation problem is relatively sparse, *i.e.* most gradient updates only affect a small portion of the parameters, it can be shown that asynchronous SGD without locking could also achieve near-optimal performance. Although this allows processors to access the shared memory with the possibility of overwriting each other's work, the final convergence is guaranteed. [22]

Chapter 6

Experiments

In this chapter, experiments are designed and carried out to verify the deep learning approaches for confidence scores by comparing with the vanilla baseline by confusion network arc posteriors. Although the effectiveness of confidence scores depend on its downstream applications, two evaluation metrics are defined to compare the performance of different models. Experiments on one-best sequences, confusion networks, and lattices are conducted sequentially. This bottom-up approach in terms of system complexity and the amount of information incorporated into the system allows us to evaluate the model progressively. Note that all numbers presented are evaluated on the test set, unless stated otherwise.

6.1 Experimental Setup

6.1.1 Data

The practicality of confidence score lies around the applications where the recognition accuracies are moderate. Therefore, for the initial investigation of applying neural networks on confidence scores, the ASR word error rate (WER) is not supposed to be too high or too low, where the distribution of confidence scores will be skewed towards either extreme. WER, the metric of the performance of the recogniser, is defined as follows.

$$\text{WER} = \frac{\text{substitution} + \text{deletion} + \text{insertion}}{\text{substitution} + \text{deletion} + \text{correct}} = 1 - \text{word accuracy} \tag{6.1}$$

One major downstream application of confidence scores is on low-resource languages, where the training data is limited for the system to reach accuracies achieved for major languages like English and Mandarin. Taking the above factors into consideration, Georgian speech corpus from the BABEL project [6] is selected as the dataset for experimentation. The

recogniser operates at around 30% WER. Its reasonable size allows fast iteration for training and debugging. Table 6.1 gives an overview of the size of the dataset and Table 6.2 shows the corresponding ASR system performance.

	training set	test set
length of speech data (hours)	17.5	7.5
number of utterances (thousand)	20.6	8.7
number of words (thousand)	103.6	45.3

Table 6.1 Georgian dataset statistics.

	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

Table 6.2 Georgian ASR system performance.

6.1.2 Evaluation Metrics

In this project, the word confidence is estimated and the following two isolated metrics are employed as general guidelines indicating how well the models perform.

- Normalised cross entropy (NCE) score. As the name suggested, this metric is based on principles of information theory that measures the amount of information gained by comparing the confidence scores obtained and a baseline where all scores equal to the probability of a word being correct in the dataset.

$$\text{NCE} = \frac{\mathcal{H}(Y^*) - \mathcal{H}(Y|\mathcal{M})}{\mathcal{H}(Y^*)} \tag{6.2}$$

  where $\mathcal{H}(Y^*)$ is the entropy of the Levenshtein distance-based labels described in § 5.1.4 and $\mathcal{H}(Y|\mathcal{M})$ is the conditional entropy of estimated confidence scores by the model $\mathcal{M}$.

The entropy of the tagged sequence is equivalent to

$$\mathcal{H}(Y^*) = -\bar{p}\log(\bar{p}) - (1-\bar{p})\log(1-\bar{p}) \tag{6.3}$$

where $\bar{p}$ is the percentage of correctly recognised word. And the conditional entropy of confidence score sequence is defined as

$$\mathcal{H}(Y|\mathcal{M}) = -\frac{1}{L}\sum_{i=1}^{L}\Big(y_i^*\log(y_i) + (1-y_i^*)\log(1-y_i)\Big) \tag{6.4}$$

where L is the total number of words in the dataset evaluated, y^* are the target labels, and y are predicted confidence scores.

If NCE is positive, then the confidence scores estimated provide some extra information than simply using the chance of a word being correct. An increase of NCE score indicates some improvement of performance.

- Area under curve (AUC) score. One of the curves for binary classification is the receiver operating characteristic (ROC) curve. ROC curve is a plot of true positive rate (TPR) against the false positive rate (FPR) at various threshold values, which illustrates the discriminative ability of the system. TPR is also known as the probability of detection, *i.e.* the proportion of words with confidence scores above the threshold that are also tagged as correct against the reference. FPR is the same as the probability of false alarm, *i.e.* the proportion of words with confidence scores below the threshold but is tagged as correct against the reference. ROC curve visualises the trade-off between rejecting the correct hypotheses and accepting the incorrect hypotheses. On such plots, curves that are above the diagonal show improvement than the baseline with the probability for all words. The goal is to push the ROC curve toward the the top-left corner, where a threshold exists such that all classification results are the same as the reference.

6.1.3 Decision Tree Baseline

In order to compare the performance of the methods proposed to estimate the confidence scores, a simple yet frequently used baseline is adopted – mapped word posterior probabilities [4]. Confusion network arc posterior probabilities normally overestimate the true confidence. Hence, a piece-wise linear and monotonically increasing function, *i.e.* a decision tree as shown in Fig. 6.1, is trained to map the word posteriors to appropriate confidence scores.

The NCE score increases noticeably whereas the AUC remains identical. This is as expected since the decision tree should map the overestimated arc posteriors closer to the true confidence

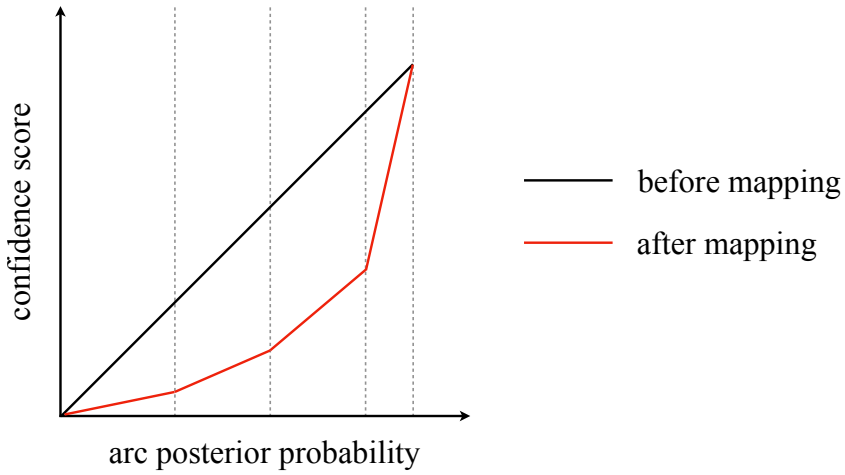


Fig. 6.1 Mapping function between arc posteriors and confidence scores.

word posteriors	NCE	AUC
before mapping	-0.1380	0.8488
after mapping	**0.2877**	0.8488

Table 6.3 Decision tree mapping of posterior probabilities.

while the total order is maintained due to the monotonicity. After mapping, the ROC curve would look exactly the same as before, but the hidden thresholds are mapped according to the decision tree. In the remaining of this chapter, the mapped confusion network arc posterior probabilities by a decision tree are regarded as the *baseline*.

6.2 Experiments

6.2.1 One-best Sequences

With the decision tree mapped confusion network posterior probabilities being the baseline for confidence scores on Babel Georgian dataset, a BLSTM model is trained using the one-best sequence to investigate the effect of multiple input features to such sequence model. During training, the standard SGD with momentum is used. For a word i in the one-best sequence, the most accessible features for include

- word identity, represented by a 50-dimensional word embedding $\mathbf{w}_i$ using fastText model described in § 5.1.2;
- time duration of the word Δt_i, which is a scalar;

- mapped confusion network arc posterior probability $\tilde{p}_i$, which is a scalar.

Therefore, the input $[\mathbf{w}_i, \Delta t_i, \tilde{p}_i]$ has 52 dimensions. The BLSTM model has 128 hidden units and the hidden states from both directions are concatenated. The output layer reduces the output dimension to 1, which is followed by a sigmoid function to produce the confidence score. The network is trained using `Hogwild!` SGD with momentum. The dataset is shuffled before the start of every epoch. The initial learning rate is set to 0.15 and the momentum is 0.5. The learning rate is reduced by half when the validation cross entropy stopped decreasing for consecutive two epochs. To prevent over-fitting, a weight decay factor of 0.001 is applied to the output layer. Parameter updates are clipped when the absolute values are greater than 1 to avoid exploding gradient, which may be observed in RNNs with long sequences.

Fig. 6.2 plots the training and validation loss during training where the fluctuations are caused by the shuffling of the dataset. In the ROC curve in Fig. 6.3, the BLSTM model is consistently closer to the top-left corner of the plot, indicating an improved performance. Table 6.4 compares the BLSTM with the baseline and shows improvement on both NCE score (7.86%) and AUC score (1.19%).

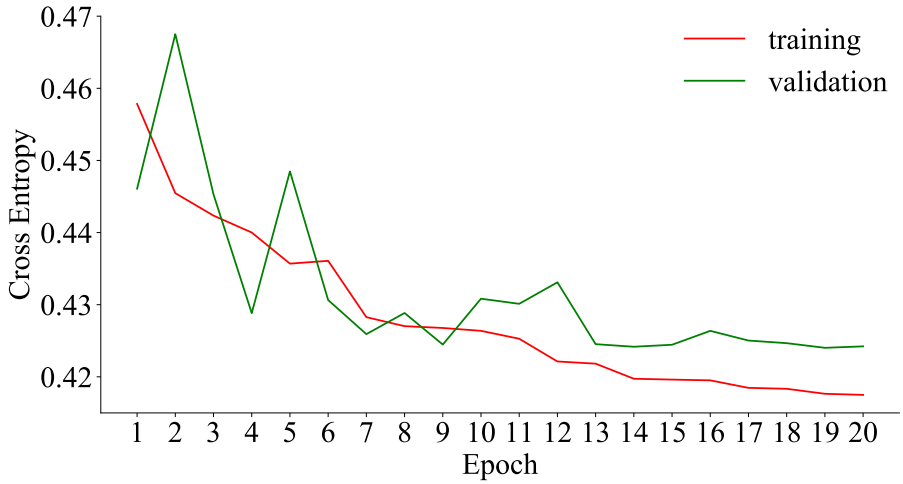


Fig. 6.2 Training plot of the BLSTM model.

	NCE	AUC
baseline	0.2877	0.8488
BLSTM	**0.3103**	**0.8588**

Table 6.4 BLSTM model improves confidence score estimation on one-best sequences.

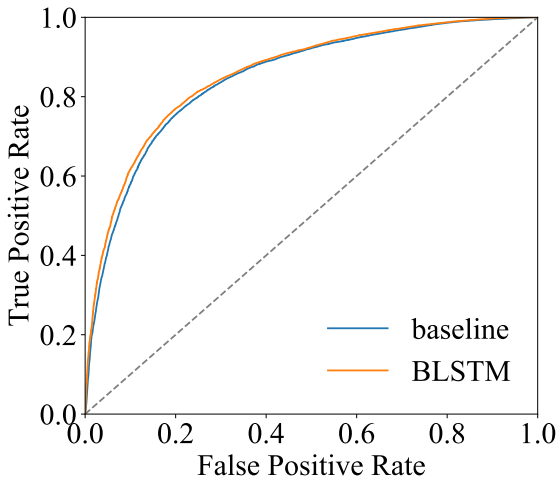


Fig. 6.3 ROC curve of confidence scores by the BLSTM model.

The improvement from the baseline to the BLSTM model is as expected. Because BLSTM not only uses the baseline confidence score as one of its features, but also incorporates the word representation and duration information to make predictions. To demonstrate the contribution of each feature, an ablation study is conducted.

model	feature vector	NCE	AUC
baseline	–	0.2877	0.8488
BLSTM	$[\mathbf{w}_i]$	0.0608	0.6664
	$[\Delta t_i]$	0.0241	0.5833
	$[\tilde{p}_i]$	0.2969	0.8530
	$[\mathbf{w}_i, \Delta t_i]$	0.0772	0.6804
	$[\Delta t_i, \tilde{p}_i]$	0.3012	0.8546
	$[\mathbf{w}_i, \tilde{p}_i]$	0.3068	0.8578
	$[\mathbf{w}_i, \Delta t_i, \tilde{p}_i]$	0.3103	0.8588

Table 6.5 An ablation study of BLSTM input features.

From Table 6.5, word embedding or duration alone does not offer much information on the word confidence. Slight gain is observed when just passing the baseline confidence scores into the bidirectional model. This is an indication that there is some sequential information between the word confidence in the one-best sequences, which is captured by the BLSTM model. Table 6.5 also shows that word embedding, duration and mapped arc posteriors are more or less complementary to each other when estimating the confidence scores. The BLSTM

model could extract useful information from not only individual features, but a combined representation of features. The ablation experiments could well demonstrate the powerful modelling capability of BLSTM model without assuming independence between multiple features. Furthermore, the model could find certain corresponding hidden representation to extract the complementary information from multiple features.

6.2.2 From One-best Sequences to Confusion Networks

Since the BLSTM model performs well with one-best sequences that are generated from confusion networks, it is of great interest to incorporate other information in the confusion network under the framework of LATTICERNN . However, there are a number of differences between one-best sequences and confusion networks. Any degradation in performance is noted and analysed carefully.

- Data:
  - The one-best sequence can be in its raw form as it is generated from the confusion network, *i.e.* the sequence may include words such as `<s>`, `</s>`, `<hes>`, `!NULL`. These words are stripped to give the final transcription.
  - As described in § 6.1.3, the word posterior probabilities are originally unmapped. They are mapped by a decision tree to compensate the overestimation as the baseline.
- Source: MLF is an HTK file format [38] containing the word transcriptions and their corresponding time frames. CN is the originally stored confusion network format. The main difference between the two is the timestamp information where the one in CN is considered to be less accurate due to clustering stages upon generation.
- Label: as described in § 5.1.4, the arcs can either be labelled in the same way the distance between two sentences is computed or be aligned using timestamp information. The threshold for time overlapping is set to $\eta_{\text{th}} = 0.1$, as it minimises the difference between labels following the Levenshtein distance and labels from the time alignment. The discrepancies between them are detailed in Table 6.6.

	Lev. $= 1$	Lev. $= 0$
time $= 1$	68.21%	0.25%
time $= 0$	0.80%	30.74%

Table 6.6 Arc tagging confusion matrix.

	data		source	label	posterior		LATTICERNN	
	sequence	posterior			NCE	AUC	NCE	AUC
O1	stripped	mapped	MLF	Lev.	0.2877	0.8488	0.3103	0.8588
O2	stripped	unmapped	MLF	Lev.	−0.1380	0.8488	0.3020	0.8529
O3			CN	Lev.	−0.1407	0.8488	0.3001	0.8524
O4			CN	time	−0.1407	0.8488	0.2879	0.8481
O5	unstripped	unmapped	CN	time	−0.1407	0.8488	0.2922	0.8492

Table 6.7 Experimental results on one-best sequences.

Table 6.7 tabulates the experimental results on different settings of one-best sequences. All NCE and AUC scores are calculated against the Levenshtein distance-based labels. Experiment O1 is identical to the one described in § 6.2.1. To directly use the information from the one-best sequences, a LATTICERNN model is trained (O2) to show that the model itself can reasonably achieve the piece-wise linear mapping by its non-linear transform of features. When moving from one-best sequences to confusion networks, the information contained in the confusion network should be used instead of the one from MLF files. Experiment O3 illustrates that the model is able to adjust the more noisy duration feature and is able to predict confidence scores with similar accuracies. From O1 to O3, the marginal degradation in performance is mainly caused by the unmapped arc posteriors. The degradation is not surprising considering that the network in O2 needs to accomplish the work done by a decision tree and the feature extraction achieved by O1. However, with fine-tuning of hyper-parameters, smarter learning rate scheduling and more iterations, LATTICERNN in O3 is expected to be level the performance of O1 despite discrepancies in input features.

The major restriction of Levenshtein label is that it is only available for one-best sequences. Moving towards confusion network, it is ideal to have a tagging scheme to effectively label all arcs in the confusion network. Experiment O4 is designed to examine the influence of timestamp-based tagging. Although the tagging accuracy w.r.t. Levenshtein distance-based labels is as high as 98.95%, the performance drops to the baseline level. It is arguably true that training criterion and testing criterion are mismatched in this case, but the damage caused by the 1% of inconsistent labels is unexpectedly large. O5 is a sanity check on the skipped operation implementation when the LATTICERNN sees special words such as `<s>`, `</s>`, `<hes>`, `!NULL`. Theoretically, the performance of O4 and O5 should be very close. The little discrepancies here could be explained by the stochastic nature of the training procedure. However, the conclusion would be more concrete if the results are the average of multiple runs with same settings but different random seeds.

6.2.3 Confusion Networks

Building upon the one-best sequences, LATTICERNN is able to incorporate more information from the confusion network and try to extract rich features from both the inputs and the special data structure. The model is expected to reach a higher level of performance when the competing hypotheses are available as a part of the data. Following experiments have been conducted to illustrate the impact of training labels and merging functions.

	data	merge	label	loss	posterior		LATTICERNN	
					NCE	AUC	NCE	AUC
O1	baseline	–	Lev.	one-best arcs	0.2877	0.8488	0.3103	0.8588
O3	one-best		Lev.		−0.1407	0.8488	0.3001	0.8524
O5	one-best		time		−0.1407	0.8488	0.2922	0.8492
C1	confnet	max	Lev.	one-best arcs	−0.1407	0.8488	0.3021	0.8532
C2			time	one-best arcs			0.2893	0.8479
C3			time	all arcs			0.2900	0.8470
C4	confnet	mean	Lev.	one-best arcs	−0.1407	0.8488	0.3018	0.8528
C5			time	one-best arcs			0.2933	0.8498
C6			time	all arcs			0.2927	0.8483
C7	confnet	posterior	Lev.	one-best arcs	−0.1407	0.8488	0.3034	0.8536
C8			time	one-best arcs			0.2939	0.8500
C9			time	all arcs			0.2914	0.8476
C10	confnet	attention	Lev.	one-best arcs	−0.1407	0.8488	0.3008	0.8527
C11			time	one-best arcs			0.2955	0.8499
C12			time	all arcs			0.2957	0.8496

Table 6.8 Experimental results on confusion networks.

This set of experiments agrees with previous findings. When transitioning from one-best sequences to confusion networks, training on timestamp-based labels results in a serious decrease in both NCE and AUC scores. For confusion networks trained on labels from the time alignment, the performance of LATTICERNN is marginally better than the baseline figures, but is still far behind the LATTICERNN model trained on the one-best sequence (O1). The training loss of Experiment C9 against the number of epochs is plotted in Fig. 6.4. A large gap between the average loss across all arcs and the average loss of all arcs on the one-best path is prominent. This is another indication of the mismatch between training and evaluation criteria. The arcs along the one-best path do not seem to be representative of all the arcs, but they are a special set of great difficulty and a major contributor to the average loss. The same

conclusion could also be reached by comparing training results of between C2 and C3, C5 and C6, *etc.* , where the final performance is virtually independent of whether the losses are accumulated only on the one-best path or on all arcs. Therefore, the following two qualities is strongly desired for an appropriate arc tagging scheme:

- labels should be nearly identical with Levenshtein distance-based labels on the one-best path;
- as far as the LATTICERNN model is concerned, the one-best path should be a representative sample from all paths in the confusion network, *i.e.* the gap of cross entropy between one-best arcs and all arcs should not be large.

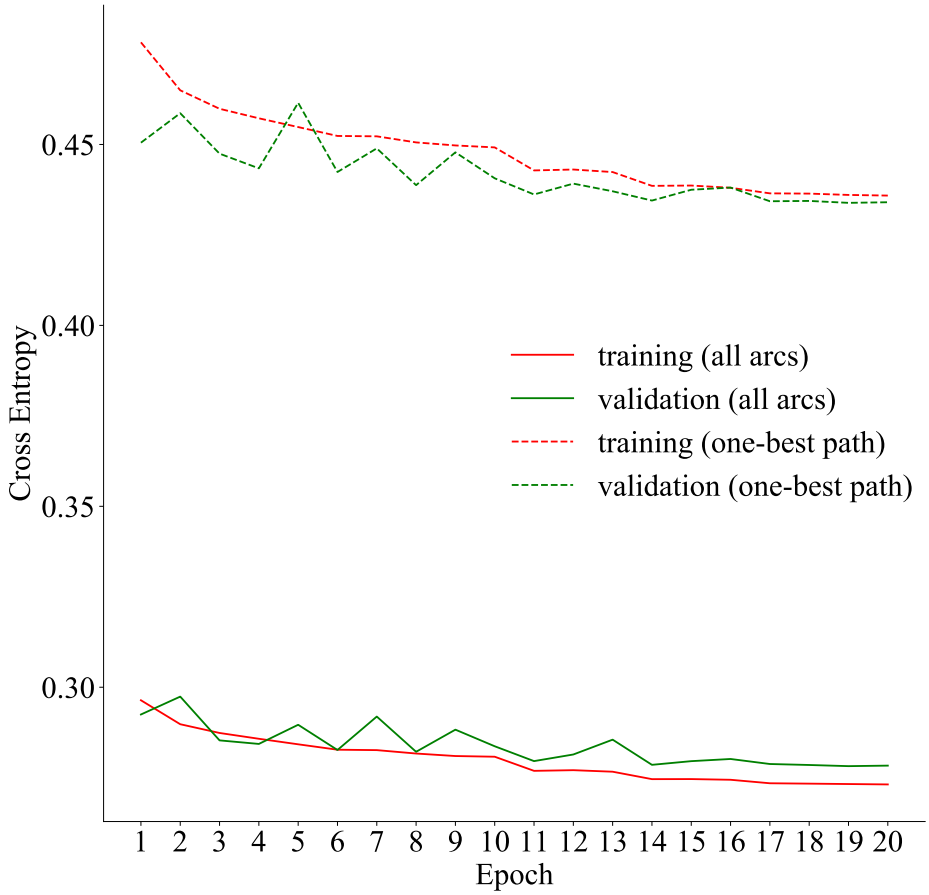


Fig. 6.4 Training plot of the LATTICERNN model on confusion networks.

Regarding different merging functions, the initial results demonstrate that posterior weighted summation is the best approach. Max function and mean function are two extreme ways of weighting, which either ignoring all other competing arcs when merging or paying too much

attention to them. Therefore, it is expected that the posterior weighted merging outperforms max function and mean function. The attention mechanism used in the experiment employs a one-layer fully connected neural network with 256 hidden units to dynamically allocate weights to different input streams. As discussed in § 4.2.2, attention mechanism is the most flexible approach for merging. Although small gain is observed by attention mechanism from C9 to C12, the design and training of the attention mechanism should be more carefully studied before reaching the conclusion that the attention mechanism is a better approach.

6.3 Discussion

All the above experiments are run under the same set of framework built using PyTorch library. It demonstrates the flexibility of the framework, which can be applied to one-best sequences (LATTICERNN reduces to BLSTM in this case), confusion networks and even general lattices produced by ASR systems. The experiments are conducted in a systematic manner where each individual step moving from one-best sequences to confusion networks is shown. Applying LATTICERNN model to sequences can improve the estimation of confidence scores significantly. Although only three basic features are used, the model is able to extract useful complementary information from all features. By incorporating more related features into the input feature vector, the LATTICERNN model is expected to have a greater gain on both NCE and AUC scores.

When the LATTICERNN model is trained on confusion networks whose arcs are labelled according to timestamp information, some degradation in performance is observed. By reconstructing the process of generating one-best sequences from confusion networks, intermediate stages are analysed. Results indicate that the arc tagging approach using time alignment is not ideal, which causes a significant drop in performance. To take full advantage of the LATTICERNN model, a better arc tagging scheme should be proposed. Nevertheless, the system and arc merging functions are shown to be working properly, with more care should be taken to design and train the attention mechanism.

Chapter 7

Conclusions and Future Work

The problem of estimating confidence scores is investigated in this project. Previous work on confidence measure is reviewed. Based on the working principles of the speech recogniser and basics of deep learning, a more general and flexible framework LATTICERNN is proposed. Implementation details are described and initial experimental results are analysed to evaluate the performance of LATTICERNN . The project serves as a proof of concept of applying the novel deep learning model in the context of confidence scores. More detailed investigations and experimentations are demanded for future work. The conclusions of this work are drawn as follows.

- Confidence score estimation is a challenging but significant task for many practical applications. Models that could incorporate the most amount of information from the speech recogniser without expert knowledge and hand-crafted features are desirable.
- The lattice-based approach is strongly favoured since the lattices contain the complete information in the speech recogniser. It could offer an in-system solution for many systems that require confidence scores if estimations could be made at the lattice level, *e.g.* unsupervised acoustic model adaptation and lattice rescoring.
- The key to confidence estimation is information extraction from all available features on structured data. Deep learning approaches are particularly useful due to the powerful non-linear modelling capability of neural networks.
- Interpretation and analysis of deep neural networks are not straightforward. Unlike forward-backward algorithm where the forward and backward messages are interpretable as probability distributions, the hidden states passing through the LATTICERNN are much less so.

- Notices should be taken on the trade-off between complexity and efficiency. Although modern computing facilitates the training of more complex models, the gain in performance may not be justified given the increased complexity under certain applications.

Based on the progress of the project, possible future work is suggested as follows.

- Detailed investigation of the arc tagging approach, as the current timestamp-based tagging may not be optimal.
- Experimentations on lattices to evaluate the performance. The lattice could be beam-pruned or UAPS-pruned, where appropriate comparisons could be made to evaluate the sensitivity of LATTICERNN on different inputs.
- Implementations to accelerate the training of LATTICERNN , especially on large lattices. Parallelism on a smaller granularity is possible. BPTS could be restricted to a certain depth as the back-propagation to root may not be worthwhile on large structures.
- Experimentations on different datasets to verify that LATTICERNN can be applied more generally.
- Incorporation of more features that could be obtained from the decoding process. Also, sub-word features may also be used, *e.g.* phone embeddings.
- Integration of LATTICERNN system to other applications, *e.g.* system combination, keyword spotting and dialogue systems.

References

[1] Tasos Anastasakos and Sreeram V. Balakrishnan. The use of confidence measures in unsupervised adaptation of speech recognizers. In *ICSLP*, 1998. 10, 16

[2] John Blatz, Erin Fitzgerald, George F. Foster, Simona Gandrabur, Cyril Goutte, Alex Kulesza, Alberto Sanchís, and Nicola Ueffing. Confidence estimation for machine translation. In *COLING*, 2004. 10

[3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *TACL*, 5:135–146, 2017. 27

[4] Gunnar Evermann and Philip C. Woodland. Large vocabulary decoding and confidence estimation using word posterior probabilities. In *ICASSP*, 2000. 10, 16, 34

[5] Gunnar Evermann and Philip C. Woodland. Posterior probability decoding, confidence estimation and system combination. In *NIST Speech Transcription Workshop*, 2000. 10

[6] Mark J. F. Gales, Kate Knill, Anton Ragni, and Shakti P. Rath. Speech recognition and keyword spotting for low-resource languages: Babel project research at cued. In *SLTU*, 2014. 32

[7] Christoph Goller and K Andreas. Learning task-dependent distributed representations by backpropagation through structure. 1996. 25

[8] Timothy J Hazen, Stephanie Seneff, and Joseph Polifroni. Recognition confidence scoring and its use in speech understanding systems. *Computer Speech & Language*, 16(1):49–67, 2002. 10, 14

[9] Geoffrey E. Hinton, Li Deng, Dong Yu, G. Dahl, A. Mohamed, Neelam K Jaitly, Andrew Senior, Vincent Vanhoucke, P. Nguyen, Tara N. Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29:82–97, 2012. 2

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. 18

[11] Hui Jiang. Confidence measures for speech recognition: A survey. *Speech Communication*, 45:455–470, 2005. 10

[12] Kaustubh Kalgaonkar, Chaojun Liu, Yifan Gong, and Kaisheng Yao. Estimating confidence scores on asr results using recurrent neural networks. In *ICASSP*, 2015. 17

[13] Simo O. Kamppari and Timothy J. Hazen. Word and phone level acoustic confidence scoring. In *ICASSP*, 2000. 14

[14] Thomas Kemp and Thomas Schaaf. Estimating confidence using word lattices. In *Eurospeech*, 1997. 13

[15] Faisal Ladhak, Ankur Gandhe, Markus Dreyer, Lambert Mathias, Ariya Rastrow, and Björn Hoffmeister. Latticernn: Recurrent neural networks over lattices. In *Interspeech*, 2016. 17, 20, 23

[16] Mingkun Li and Ishwar K. Sethi. Confidence-based active learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28:1251–1261, 2006. 10

[17] Lidia Mangu, Eric Brill, and Andreas Stolcke. Finding consensus among words: lattice-based word error minimization. In *Eurospeech*, 1999. 12

[18] Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013. 27

[19] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013. 27

[20] Julian James Odell. *The Use of Content in Large Vocabulary Speech Recognition*. PhD thesis, Cambridge University Engineering Department, 1995. 7

[21] Joel Pinto and R. N. V. Sitaram. Confidence measures in speech recognition based on probability distribution of likelihoods. In *Interspeech*, 2005. 14

[22] Benjamin Recht, Claudia Ernestina Re, Stephen J. Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 2011. 31

[23] Alberto Sanchis, Alfons Juan, and Enrique Vidal. Estimating confidence measures for speech recognition verification using a smoothed naive bayes model. In *Iberian Conference on Pattern Recognition and Image Analysis*, 2003. 14

[24] Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45:2673–2681, 1997. 19

[25] Matthew Seigel. *Confidence Estimation for Automatic Speech Recognition Hypotheses*. PhD thesis, Cambridge University Engineering Department, 2013. 2, 15

[26] Matthew Stephen Seigel and Philip C. Woodland. Combining information sources for confidence estimation with crf models. In *Interspeech*, 2011. 15

[27] Jinsong Su, Zhixing Tan, Deyi Xiong, Rongrong Ji, Xiaodong Shi, and Yang Liu. Lattice-based recurrent neural network encoders for neural machine translation. In *AAAI*, 2017. 17

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017. 24

[29] Mitch Weintraub, Françoise Beaufays, Zeév Rivlin, Yochai Konig, and Andreas Stolcke. Neural-network based measures of confidence for word recognition. In *ICASSP*, 1997. 14

[30] BP Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420, 1962. 28

[31] Paul J. Werbos. Backpropagation through time: What it does and how to do it. 1990. 18

[32] Frank Wessel, Ralf Schlüter, Klaus Macherey, and Hermann Ney. Confidence measures for large vocabulary continuous speech recognition. *IEEE Transactions on Speech and Audio Processing*, 9:288–298, 2001. 2

[33] Gethin Williams and Steve Renals. Confidence measures from local posterior probability estimates. *Computer Speech & Language*, 13:395–411, 1999. 14

[34] Jay G. Wilpon, Lawrence R. Rabiner, Chin-Hui Lee, and E. R. Goldman. Automatic recognition of keywords in unconstrained speech using hidden markov models. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 38:1870–1878, 1990. 10

[35] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, 2015. 24

[36] Su-Youn Yoon, Mark Hasegawa-Johnson, and Richard Sproat. Automated pronunciation scoring using confidence scoring and landmark-based svm. In *Interspeech*, 2009. 10

[37] Sheryl R. Young. Recognition confidence measures: Detection of misrecognitions and out-of-vocabulary words. In *ICASSP*, 1994. 10

[38] SJ Young, G Evermann, MJF Gales, D Kershaw, G Moore, JJ Odell, DG Ollason, D Povey, V Valtchev, and PC Woodland. The htk book version 3.4. 2006. 27, 38

[39] Torsten Zeppenfeld, Michael Finke, Klaus Ries, Martin Westphal, and Alexander H. Waibel. Recognition of conversational telephone speech using the janus speech engine. In *ICASSP*, 1997. 14