Active and Semi-Supervised Learning for Speech Recognition

Florian L. Kreyssig

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

Emmanuel College March 2023

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 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Florian L. Kreyssig
March 2023

Abstract

Active and Semi-Supervised Learning for Speech Recognition

Florian L. Kreyssig

Recent years have seen significant advances in speech recognition technology, which can largely be attributed to the combination of the rise in deep learning in speech recognition and an increase in computing power. The increase in computing power enabled the training of models on ever-expanding data sets, and deep learning allowed for the better exploitation of these large data sets.

For commercial products, training on multiple thousands of hours of transcribed audio is common practice. However, the manual transcription of audio comes with a significant cost, and the development of high-performance systems is typically limited to commercially viable tasks and languages. To promote the use of speech recognition technology across different languages and make it more accessible, it is crucial to minimise the amount of transcribed audio required for training. This thesis addresses this issue by exploring various approaches to reduce the reliance on transcribed data in training automatic speech recognition systems through novel methods for active learning and for semi-supervised learning.

For active learning, this thesis proposes a method based on a Bayesian framework termed NBest-BALD. NBest-BALD is based on Bayesian Active Learning by Disagreement (BALD). NBest-BALD selects utterances based on the mutual information between the prediction for the utterance and the model parameters, *i.e.* $\mathcal{I}[\theta, \mathrm{w}|\mathcal{D}_l, \mathbf{X}_i]$. Monte-Carlo Dropout is used to approximate sampling from the posterior of the model parameters and an N-Best list is used to approximate the entropy over the hypothesis space. Experiments on English conversational telephony speech showed that NBest-BALD outperforms random sampling and prior active learning methods that use confidence scores or the NBest-Entropy as the informativeness measure. NBest-BALD increases the absolute Word Error Rate (WER) reduction obtained from selecting more data by up to 14% as compared to random selection.

Furthermore, a novel method for encouraging representativeness in active data selection for speech recognition was developed. The method first builds a histogram over the lengths of the utterances. In order to select an utterance, a word length is sampled from the histogram, and the utterance with the highest informativeness within the corresponding histogram bin is chosen. This ensures that the selected data set has a similar distribution of utterance lengths to the overall data set. For mini-batch acquisition in active learning on English conversational telephony speech, the method significantly improves the performance of active learning for the first batch. The histogram-based sampling increases the absolute WER reduction obtained from selecting more data by up to 57% as compared to random selection and by up to 50% as compared to an approach using informativeness alone.

A further contribution to active learning in speech recognition was the definition of a cost function, which takes into account the sequential nature of conversations and meetings. The level of granularity at which data should be selected given the new cost function was examined. Selecting data on the utterance-level, as fixed-length chunks of consecutive utterances, as variable-length chunks of consecutive utterances and on the side-level were examined. The cost function combines a Real-Time Factor (RTF) for the utterance length (in seconds) with an overhead for each utterance (t_1) and an overhead for a chunk of consecutive utterances (t_2). The overhead t_2 affects the utterance-level selection method (which previous methods in the literature rely on) the most, and this level of granularity yielded the worst speech recognition performance. This result showed that it is crucial to focus on methods for selection that can take a better cost function into account.

For semi-supervised learning, the novel algorithm cosine-distance virtual adversarial training (CD-VAT) was developed. Whilst not directed at speech recognition, this technique was inspired by initial work towards using consistency-regularisation for speech recognition. CD-VAT allows for semi-supervised training of speaker-discriminative acoustic embeddings without the requirement that the set of speakers is the same for the labelled and the unlabelled data. CD-VAT is a form of consistency-regularisation where the supervised training loss is interpolated with an unsupervised loss. This loss is the CD-VAT-loss, which smoothes the model's embeddings with respect to the input as measured by the cosine-distance between an embedding with and without adversarial noise. For a large-scale speaker verification task, it was shown that CD-VAT recovers 32.5% of the Equal Error Rate (EER) improvement that would be obtained when all speaker labels are available for the unlabelled data.

For semi-supervised learning for speech recognition, this thesis proposes two methods to improve the input tokenisation that is used to derive the training targets that are used in masked-prediction pre-training; a form of self-supervised learning. The first method is biased self-supervised learning. Instead of clustering the embeddings of a model trained using unsupervised training, it clusters the embeddings of a model that was finetuned for a small number of updates. The finetuning is performed on the small amount of supervised data that is available in any semi-supervised learning scenario. This finetuning ensures that the self-supervised learning task is specialised towards the task for which the model is supposed to be used. Experiments on English read speech showed that biased self-supervised learning can reduce the WER by up to 24% over the unbiased baseline. The second method replaces the K-Means clustering algorithm that was previously used to tokenise the input with a Hidden Markov Model (HMM). After training, the tokenisation of the input is performed using the Viterbi algorithm. The result is a tokenisation algorithm that takes the sequential nature of the data into account and can temporally smooth the tokenisation. On the same English read speech task, the HMM-based tokenisation reduces the WER by up to 6% as compared to the tokenisation that uses K-Means.

Acknowledgements

I would like to express my deepest gratitude to the people who have supported me throughout my academic journey and thus contributed to the completion of this thesis.

First and foremost, I owe an immense debt of gratitude to my supervisor Prof Phil Woodland, who, nearly seven years ago, took me on as an intern and later as a PhD student. Without the opportunity to work with Phil as an undergraduate researcher, my life's journey would likely look very different. Phil's guidance, insightful discussions, and teaching on how to do research have been invaluable to me as well as his various cycles of feedback for my publications, presentations, and this thesis. I am also grateful to Phil for helping me secure the EPSRC funding that supported me throughout my PhD.

I would like to thank Dr Chao Zhang, whose continuous help throughout my entire research career has been instrumental in shaping my work as well as my thinking about research. Chao has offered me an abundance of helpful tips, advice, and feedback.

I am also grateful to Dr Qiujia Li, who has created a stimulating working environment in the office and provided me with countless ideas to develop my research further. I will always look back fondly on our short but fruitful direct collaboration on DNC.

Prof Mark Gales has been a great advisor, offering guidance and constructive criticism at every stage of my research as well as giving me the initial motivation to do research on active learning. My collaborators at Meta, including Dr Yangyang Shi, Dr Abdelrahman Mohamed, Dr Jinxi Guo, and Dr Leda Sari, have provided me with valuable contributions and fruitful discussions.

I would like to express my gratitude to Dr Guangzhi Sun, Xianrui Zheng, Dr Bo-Hsiang Tseng, Dr Yiting Lu, Yassir Fathullah, Dr Qingyun Dou, Dr Kate Knill, and Dr Linlin Wang for making the Engineering department a fun place to work at.

I would like to extend my heartfelt thanks to my wife Tiansi Jing, who has been a constant source of encouragement, love, and support. Her unwavering support and belief in me have been crucial in completing this thesis despite the challenges of our long-distance relationship.

Finally, I would like to express my gratitude to my parents and my sister, who have always believed in me, encouraged me, and supported me throughout my life. Their love, guidance, and sacrifices have been my driving force.

Table of contents

List of figures

List of tables

Notation

Statistics

$D[P_{\theta_1}(y), P_{\theta_2}(y)]$	distance between two distributions *e.g.* the KL-divergence
$\mathbb{H}[A]$/$\mathbb{H}_{p(A)}[A]$	entropy of A / evaluated under distribution $p(A)$
$\mathbb{E}[A]$/$\mathbb{E}_{p(A)}[A]$	expectation of A / evaluated under distribution $p(A)$
$\mathcal{N}(\mu, \sigma^2)$	univariate normal distribution with mean μ and variance σ^2
$P(\cdot)$	probability mass function
$P_\theta(\cdot)$	probability mass function, parameterised by θ
$p(\cdot)$	probability density function
$p_\theta(\cdot)$	probability density function, parameterised by θ
$\mathcal{U}(a, b)$	uniform distribution in the range $[a, b]$

Learning and Data

α	interpolation coefficient for losses (as used in Chapter 6)
$\mathcal{C}/\mathcal{C}_{\text{S}}/\mathcal{C}_{\text{US}}$	general/supervised/unsupervised cost function
$\mathcal{D}_l = \{(\mathbf{X}_i; \hat{\mathbf{w}})\}_{i=1}^{N}$	labelled data set
$\mathcal{D}_{ul} = \{\mathbf{X}_i\}_{l=N+1}^{N+U}$	unlabelled data set
$\mathcal{D}_{\text{text}}$	total text corpus that is used for language model training
$\mathbf{H}$	Hessian matrix
$\mathcal{L}$	loss function
N	number of (labelled) training examples
$\mathcal{R}$	regularisation loss *e.g.* $\mathcal{R}_{\text{VAT}}(\mathcal{D}_l, \mathcal{D}_{ul}, \theta)$
U	number of unlabelled training examples
$\theta/\theta^*/\theta'$	trainable/optimal/fixed model parameters
$\Delta\theta$	parameter update

Neural Networks

$a, \mathbf{a}$	linear pre-activation value/vector
B	minibatch size
$b, \boldsymbol{b}$	bias value/vector
$(D, D^{(0)}), D^{(l)}$	dimensionality of the input, number of nodes in layer l

$e, \boldsymbol{e}, \boldsymbol{E}$	encoding/embedding value/vector/matrix
$f(\cdot), g(\cdot)$	generic functions
$\boldsymbol{f}, \boldsymbol{i}, \boldsymbol{o}, \tilde{\boldsymbol{c}}, \boldsymbol{c}$	forget/input/output/proposed-cell/cell values for LSTM
$h, \boldsymbol{h}$	hidden value/vector
i, j	generic indices
J	number of Monte-Carlo samples in BALD
$\boldsymbol{k}$	key vector input for attention mechanism
k, K	class index, number of classes
l	layer index
L	depth of the ANN
$\boldsymbol{M}$	trained weight matrix (for dropout explanation)
$\boldsymbol{m}$	dropout mask vector
$p^{(l)}$	layer-specific dropout rate
$q(\theta)$	variational approximation to the posterior
$s, \boldsymbol{s}$	recurrent state value/vector
t, m	time index (input,generic)/(output)
u	update index
T, M	total number of time steps (input,generic)/(output)
$\boldsymbol{v}$	momentum vector
$\boldsymbol{V}$	recurrent weight matrix
$w_{ij}, \boldsymbol{w}_i, \boldsymbol{W}$	weight value/vector/matrix
$x, \mathbf{x}, \mathbf{X}$	input value/vector/matrix
$\mathcal{X}$	domain of function
$y, \mathbf{y}, \mathbf{Y}$	output value/vector/matrix
$\mathcal{Y}$	codomain of function
$\hat{y}_i, \hat{\mathbf{y}}_i$	output label value/vector
$z, \boldsymbol{z}$	logit value/vector
α_t	attention value giving weight to vector at t
$\delta(\cdot)$	continuous Dirac delta function
η	learning rate
γ	gain factor for weight initialisation
λ	weight-decay parameter
ω	momentum parameter
$\phi(\cdot)$	activation function
$\varphi(\cdot)$	neural network, building block
$\sigma(\cdot)$	sigmoid activation function

$\texttt{Softmax}(\cdot)$	Softmax activation function
τ	learning rate decay constant
$\odot$	element-wise multiplication
Speech Recognition	
a_{ij}	probability of transitioning to state j when being in state i
$b_i(\mathbf{x}_t)$	probability of emitting $\mathbf{x}_t$ when in state i
$\boldsymbol{c}_m$	context vector at output step m
c	latent variable representing the mixture component of a GMM
Δ	size of future/past context for the MLP input
$\boldsymbol{e}, \boldsymbol{g}, \boldsymbol{h}$	embeddings
f_{mel}/f_{Hz}	frequency in the mel-scale/Hertz-scale
$\mathcal{F}$	auxiliary function using in the FB algorithm
K	number of Gaussian mixture components
$\pi_{ik}, \boldsymbol{\mu}_{ik}, \Sigma_{ik}$	GMM parameters of mixture k of logical state i
K	number of mixture components
Q	number of sub-word units
q	sub-word unit (*e.g.* phone)
$\mathbf{q}$	sequence of sub-word units
V	Vocabulary
$\mathcal{S}$	set of possible state sequences
S	total number of HMM states
S, D, I, C, M	number of substitutions, deletions, insertions, correct words, words in reference
s_t	HMM state at time t
T	Length of input sequence
Θ	delta window for speech input features
$w, \mathbf{w}$	word, word sequence
$\mathbf{w}^\star$	recognised output word sequence / hypothesis
$\hat{\mathbf{w}}$	correct word sequence
$\alpha_{m,t}$	attention weight between input time-step t and output step m
$\mathcal{B}^{-1}(\mathbf{w})$	set of neural transducer/ alignments that are consistent with $\mathbf{w}$
ι	insertion penalty (negative-valued)
κ	grammar scaling factor
ω	attention coverage coefficient
$\psi_j(t)$	maximum likelihood of sequences ending with $s_t = j$
τ	attention coverage threshold
$\emptyset$	blank unit in CTC modelling

Speaker Verification and CD-VAT (Chapters 4 and 6)

$\mathbf{c}_i$	centroid of the embeddings of speaker i
$\mathrm{cd}[\mathbf{a},\mathbf{b}]$	cosine distance between $\mathbf{a}$ and $\mathbf{b}$
$D(\mathbf{r},\mathbf{x},\theta)$	shorthand for $D[P_{\theta'}(y \mid \mathbf{x}), P_\theta(y \mid \mathbf{x}+\mathbf{r})]$
$\mathbf{u}/\mathbf{u}(\mathbf{x},\theta)$, $\lambda_1(\mathbf{r},\mathbf{x},\theta)$	dominant eigenvector of $\mathbf{H}$ and its eigenvalue
$\mathbf{e}(\mathbf{x},\theta)$, $\mathbf{e}$	embedding generated for input $\mathbf{x}$
$\mathcal{E}_i$	All embeddings belonging to speaker i
ϵ	Size of input perturbation
$\mathbf{g}_i$	gradient at iteration i of calculating the adversarial noise w.r.t to the input of the distance between model's output with and without input noise $\zeta\mathbf{v}_{i-1}$
$\mathbf{H}/\mathbf{H}(\mathbf{x},\theta)$	shorthand for $\nabla\nabla_{\mathbf{r}}D(\mathbf{r},\mathbf{x},\theta)\mid_{\mathbf{r}=\mathbf{0}}$
I	the number of power iterations when calculating the adversarial noise
$\mathrm{LCS}(\mathbf{x},\theta)$	local cosine smoothness for input $\mathbf{x}$ and parameters θ
$\mathrm{LDS}(\mathbf{x},\theta)$	local distribution smoothness for input $\mathbf{x}$ and parameters θ
$\mathbf{r}$	Input perturbation
$\mathbf{v}_i$	L_2-normalised input perturbation during iteration i of calculating the adversarial noise $\mathbf{r}_{\text{VAT}}$. $\mathbf{v}_0$ is sampled from a Gaussian distribution.
$\mathbf{r}_{\text{VAT}}/\mathbf{r}_{\text{CDVAT}}$	virtual adversarial noise for VAT and CD-VAT
S	the number of speakers
ϕ/ϕ_{EER}	threshold for scoring/ at which EER is achieved
ζ	Size of input noise used for $\mathbf{r}_{\text{CDVAT}}$ calculation

Active learning and NBest-BALD (Chapters 5, 7 and 8)

F_i	Partion of data
λ_k	the k-th feature used to calculate the TF-IDF vector
t_1	gap left between automatically segmented utterances during transcription
t_2	overhead of transcription for a chunk of consecutive utterances
$\mathtt{RTF}$	RTF used to calculate transcription cost
$\mathtt{sim}(\mathbf{X}_j,\mathbf{X}_i)$	Similarity between utterances $\mathbf{X}_j$ and $\mathbf{X}_i$
$\mathrm{tf}(\lambda_k,\mathbf{X}_i)$	term frequency (TF) of feature λ_k in utterance $\mathbf{X}_i$
$\mathrm{idf}(\lambda_k)$	inverse document frequency (IDF) of feature λ_k
$\mathbf{m}(\mathbf{X}_i)$	the TF-IDF vector for utterance $\mathbf{X}_i$

Masked prediction pre-training (Chapter 9)

α	number of MPPT updates, *e.g.* 100k for 100k updates
β	number of clusters used for the tokenisation in , *e.g.* 100 for $K = 100$
c_t,l_t	cluster-label and ground-truth label of vector $\mathbf{x}_t$
K, L	number of clusters, number of ground-truth labels

I	number of iterations of MPPT
$\mathcal{M}_i$	model trained during the i-th iteration of MPPT
C_{β}^{α}	model ID for model trained using , embeddings of M_{100}^{250k} after the biasing step were clustered for tokenisation
H_{β}^{α}	model ID for model trained using , embeddings M_{100}^{250k} were clustered for tokenisation
M_{β}^{α}	model ID for model trained using , MFCCs were clustered for tokenisation
S_{β}^{α}	model ID for model trained using , embeddings of C_{100}^{250k} after the biasing step were clustered for tokenisation
$\mathcal{T}$	tokenisation used in MPPT

Acronyms

A | B | C | D | E | F | G | H | I | K | L | M | N | O | P | Q | R | S | T | V | W

A

AED Attention-Based Encoder-Decoder. 34, 109
AM Acoustic Model. 34, 37–39, 47–49, 52, 53, 57, 61, 62, 65, 86
ANN Artificial Neural Network. 7, 8, 10, 13, 16, 17, 20–29, 34, 35, 37, 45, 47, 48, 52, 62, 64, 68, 71, 81, 83, 84, 86
ASR Automatic Speech Recognition. 1–5, 7, 17, 19, 29, 31, 33–35, 49, 50, 56, 57, 60–62, 65, 66, 68–71, 74–76, 78, 80, 81, 83, 85, 86, 88, 89, 91, 93, 94, 96, 98, 100, 102–104, 107, 109, 121, 124, 128, 129, 133–135, 137, 140, 144, 147, 149, 151, 153–158, 161, 165, 166, 168, 170, 174, 175, 177, 179–181

B

BALD Bayesian Active Learning by Disagreement. v, 98, 123, 124, 126
BERT Bidirectional Encoder Representations from Transformers. 86, 87
BPE Byte Pair Encoding. 38, 162

C

CD-VAT cosine-distance virtual adversarial training. vi, xiv, 4, 5, 89, 104, 112, 114–119, 177, 178, 180
CE Cross Entropy. 21, 64, 65, 68, 70, 77, 104, 106–109, 118, 128, 166, 167, 175, 178, 180
CH CallHome. 76, 127, 130–133, 140, 143, 144, 184, 185
CML Conditional Maximum Likelihood. 21
CNN Convolutional Neural Network. 13, 15, 86, 87
CTC Connectionist Temporal Classification. 49, 66, 67, 71, 99, 155, 158, 161, 162

D

DCT Discrete Cosine Transform. 36

DET Detection Error Tradeoff. 117
DFT Discrete Fourier Transform. 35, 36
DNN Deep Neural Network. 1, 46, 83, 104, 107, 108, 116, 118, 128, 167, 178
DSP digital signal processing. 33

E

EER Equal Error Rate. vi, 110, 111, 116–118, 178
EGL expected gradient length. 99
EM Expectation Maximisation. 62, 63, 81, 154

F

FBANK Filter Bank. 35, 36, 71, 86, 107
FC fully-connected. 10, 13–15, 22, 128
FNN Feedforward Neural Network. 10
FSH Fisher. 185

G

GMM Gaussian Mixture Model. 1, 11, 35, 37, 45, 47, 48, 62–66, 71, 76, 168
GRU Gated Recurrent Unit. 15

H

HMM Hidden Markov Model. vii, xvii, 1, 3–5, 34, 35, 37, 39–42, 44–49, 52–55, 62–68, 70, 71, 100, 101, 104, 107–109, 116, 118, 154, 167–171, 173–175, 178, 180, 181
HuBERT Hidden unit BERT. 88, 153–155, 158, 160, 161

I

ISC intra-speaker compactness. 111, 112, 114, 117, 118, 178
ISS inter-speaker separability. 111, 114, 117, 118, 178

K

KL-divergence Kullback-Leibler divergence. 82, 83, 98, 103, 105–107, 122

L

LF-MMI Lattice-Free Maximum Mutual Information. 48, 49, 65, 134, 136
LM Language Model. 34, 49–55, 57, 59, 65, 66, 85, 125, 128, 185
LSTM Long Short-Term Memory. 15, 16, 25

M

MAP Maximum a Posteriori. 27
MAPSSWE Matched-Pair Sentence-Segment Word Error. 132–135, 143, 150
MBR Minimum Bayes Risk. 60
MFCC Mel-scale Frequency Cepstral Coefficient. 36, 45, 86, 88, 114, 155
ML Maximum Likelihood. 21, 26–28, 50, 52, 53, 62, 65, 70
MLP Multilayer Perceptron. 9, 10, 13, 17, 19, 31, 47, 69, 73, 75
MMI Maximum Mutual Information. 65, 66, 80, 81
MPE Minimum Phone Error. 81
MPPT Masked-Prediction Pre-Training. 153–155, 158, 159
MSE Mean Squared Error. 21, 22, 106

N

NLP Natural Language Processing. 85, 86
NNLM Neural Network Language Model. 67

O

OOV Out-of-Vocabulary. 102

P

PLP Perceptual Linear Prediction. 36, 45, 71

Q

QBC Query-By-Committee. 98, 100

R

RBM Restricted Boltzmann Machine. 83
RNN Recurrent Neural Network. 7, 13, 15–17, 23, 47, 48, 51, 57, 61, 68, 69, 85
RTF Real-Time Factor. vi, 145–147, 151, 179

S

SDS Spoken Dialogue System. 29, 61
SGD Stochastic Gradient Descend. 23, 25, 27, 28, 115
SVM Support Vector Machine. 83
SWB SwitchBoard. 76, 121, 127, 130–133, 138, 140, 143, 144, 184, 185

T

TDNN Time Delay Neural Network. 13–15, 31, 47, 48, 87, 108, 115, 133

TF-IDF term frequency inverse document frequency. 100, 126, 127, 130, 135, 181

V

VAE Variational Auto-Encoder. 83

VAT virtual adversarial training. 4, 82, 89, 103–105, 107–109, 112, 117, 118, 178, 181

W

w.r.t with respect to. 62, 103, 105, 178

WER Word Error Rate. v–vii, 56, 57, 60, 92, 94–96, 130–135, 138, 140, 143, 144, 148–152, 156–158, 162–165, 167, 173–175, 179, 180, 184

WFST Weighted Finite State Transducer. 55, 56

Chapter 1

Introduction

For a long time, humans have harboured the dream of conversing with machines. Automatic Speech Recognition (ASR) technology has made this dream a reality by enabling computers to convert human speech to text, making speech-based interaction with machines possible. The integration of ASR has become increasingly important in numerous software applications, particularly on devices lacking a proper keyboard. Personal intelligent assistants, such as Apple Siri or Amazon Alexa, have significantly transformed the way people access information and services compared to a decade ago. These assistants rely on speech input, and the growing popularity of such devices is evident in the increasing demand for speech-related services like telephony transcription, meeting transcription, and video captioning. Driven by practical applications, various speech processing systems are constantly improving in performance.

ASR is the mapping from the spoken audio waveform to text. This text can then be processed by a downstream application. Modern methods for ASR focus on statistical approaches that model the probability distribution of how probable a word sequence $\mathbf{w}$ is given the (processed) input audio sequence $\mathbf{X}$ *i.e.* $P(\mathbf{w} \mid \mathbf{X})$. Multiple options exist to model this distribution. A popular family of approaches models the distribution using a combination of a Hidden Markov Model (HMM) to model the distribution $p(\mathbf{X} \mid \mathbf{w})$ (called the acoustic model) and an n-th order Markov model to model the distribution $P(\mathbf{w})$ (called the language model). The HMM models the acoustic input using Gaussian Mixture Models (GMMs) or Deep Neural Networks (DNNs). Another family of approaches directly models $P(\mathbf{w} \mid \mathbf{X})$ through a single neural network that can be trained as a whole.

Speech recognition technology has seen major improvements in recent years. Partly due to using DNNs, this has been mainly driven by increases in computing power, leading to the possibility of training on ever-larger data sets. For commercial products, it is normal to use thousands of hours of transcribed audio for training. Manually transcribing audio has a significant cost associated with it and therefore high-performance systems are built for the specific tasks and languages that are commercially viable. For speech recognition technology to be more prevalent in all parts of life and to democratise the technology over various languages, it is important to reduce the amount of transcribed audio needed for training. This thesis focuses on techniques that help to reduce the amount of transcribed data that is needed for ASR training. The work discussed in this thesis focuses on two research areas for reducing the amount of needed transcribed data: *semi-supervised* learning and *active* learning.

In this chapter, semi-supervised learning and active learning are first briefly introduced. The thesis outline is presented, and the main novel contributions are listed.

1.1 Semi-supervised learning

Standard training of an ASR system is done via supervised learning on a supervised training set. The training set contains labelled utterances, meaning many audio sequences and their corresponding word sequences. These word sequences were obtained by humans who manually transcribed the data. This data set contains the examples that are used to learn the distribution $P(\mathbf{w}|\mathbf{X})$. In a semi-supervised learning setting, the speech recogniser and its learning algorithms have access to both such a supervised data set and to a (usually much larger) unsupervised data set, which is a corpus of utterances for which only the audio sequences are available. One of the most common approaches to semi-supervised learning of ASR systems is self-training, where a seed system trained with only supervised data is used to recognise the unsupervised data, and the predicted hypotheses are selected as the new training transcripts. This enlarges the training set, which can then be used for supervised training. Recently popularised approaches are focused on using unsupervised representation learning (using the unsupervised data set) followed by supervised finetuning (using the supervised data set). Here, first, a neural network is trained using the unsupervised data set to learn a good and efficient representation of the audio signal. Then, this neural network is used as part of an ASR system, and the entire system is finetuned using the supervised data set. This can be followed by self-training to improve the system further.

1.2 Active learning

The starting point of active learning is the same as for semi-supervised learning. Both a supervised and an unsupervised data set are available. The difference is that a budget is available that can be used to manually transcribe a portion of the unsupervised data set. When applying active learning in an industrial setting, this budget is literally an amount of money that can be used to pay for the transcription service. In exploratory research settings, a specific number of utterances or a specific number of seconds has been used as an approximation to the real cost. The task of active learning is to choose a subset of utterances from this unsupervised data set that will improve the speech recogniser the most. The simplest baseline use for comparison would be to choose a subset randomly. Common approaches for active learning for ASR systems focus on training a seed system with only the supervised data and then selecting the set of utterances on which the ASR system is most uncertain during the recognition process. The function that scores an utterance (here by measuring the uncertainty) is called an acquisition function.

Both semi-supervised learning and active learning play a critical role in reducing the overall budget needed to build ASR systems. Progress in these areas should help accelerate the currently restricted progress of ASR technology in languages and domains that do not attract the same level of labelling investment as the major language (*e.g.* English and Mandarin Chinese). Even though it is not investigated in this thesis, given that the operating point in terms of available data is the same for both semi-supervised and active learning, it is natural to combine the two methods in order to exploit the available data further.

1.3 Thesis Outline

This thesis consists of ten chapters that are structured as follows.

- Chapter 2 first establishes the fundamentals of deep learning, including the basic building blocks such as the different neural network layers and activation functions. It introduces cost functions for training neural networks as well as associated regularisation techniques. Further, it is explained how the parameters of neural networks are learned using the backpropagation algorithm. The chapter ends with how uncertainty can be represented in neural networks.

- Chapter 3 describes ASR, the features that are used as the input, methods for acoustic modelling with a focus on HMM based modelling, language models and decoding procedures. It is outlined how ASR models can be trained, how they are evaluated, how the output of an ASR-model can be represented and how the confidence of the model can be modelled. Finally, end-o-end trainable ASR methods, which are Neural Transducers and Attention-based Encoder-Decoder models are described.

- Chapter 4 describes various semi-supervised learning methods for general machine learning and for ASR. These include self-training, lightly-supervised training, probabilistic generative models, cluster-then-label approaches, consistency regularisation and graph-based methods. The chapter ends with methods for unsupervised representation learning that recently became the most popular set of algorithms.

- Chapter 5 focuses on the background around active learning. It describes the various categories of acquisition functions and various examples thereof. Further, an introduction to the labelling cost in ASR is given.
- Chapter 6 focuses on consistency regularisation for speech processing based on virtual adversarial training (VAT). It can be split into a *negative* result concerning consistency regularisation for ASR and a *positive* result concerning consistency regularisation for learning speaker embeddings in the form of a novel, proposed loss function called cosine-distance virtual adversarial training (CD-VAT). CD-VAT is shown to be effective on a speaker verification task.
- Chapter 7 develops a Bayesian active learning approach for ASR by adjusting an algorithm that was originally developed for machine learning tasks with a single output to the sequence-to-sequence task of ASR. The developed technique termed NBest-BALD is shown to be effective on a conversational telephone speech task.
- Chapter 8 focuses on two innovations for active learning in ASR. The first fixes a long-standing issue of active learning for ASR which is that too many short utterances are selected. The proposed algorithm matches the distributions over the utterance lengths of the unsupervised data set and the selected subset. The second innovation is that previously published active learning algorithms for ASR do not take the nature of the cost of transcribing into account, especially in the case of meetings or conversations. A new definition of the transcription budget is developed, and a method for selecting groups of consecutive utterances is presented.
- Chapter 9 focuses on self-supervised representation learning for ASR. Previous algorithms separate the ASR training into an unsupervised pre-training stage and a supervised fine-tuning stage. The chapter introduces *biased self-supervised training* where the supervised data that is used in the finetuning stage is introduced into the self-supervised pre-training stage. This method is shown to be highly effective both in terms of the final performance, but also in terms of the training speed. Further, a method to use an HMM to derive the targets for self-supervised learning is proposed and compared to a previous clustering-based approach.

- The key conclusions and contributions are summarised in Chapter 10. Based on the current findings and the understanding of active and semi-supervised learning, Chapter 10 also suggests a number of related topics for further investigation.

1.4 Contributions

The key novel contributions of this thesis are as follows.

1. A technique for semi-supervised learning of speaker embeddings termed cosine-distance virtual adversarial training (CD-VAT) is proposed. The method consists of a regularisation loss that is calculated using the unsupervised data set. The loss ensures that the model is smooth with respect to the input on the data manifold. CD-VAT is evaluated and shown to be effective for speaker verification. This was published in Kreyssig and Woodland (2020).

2. A technique for active learning in ASR based on Bayesian principles. The particular implementation is termed *NBest-BALD*. It uses the mutual information between the prediction for the utterance and the model parameters as the selection criterion.

3. A technique to match the distribution over the utterance lengths for the untranscribed utterance pool and the utterance pool selected by the active learning algorithm. This method alleviates the issue that active learning methods for ASR tend to select shorter utterances.

4. A proposal for the use of a realistic cost function for labelling utterances and methods for selecting variable-length groups of utterances that occur in sequence in a conversation or a meeting.

5. A self-supervised learning technique that combines supervised and unsupervised data. It is analysed how this method leads to not only better performance but also faster training. This work was published in Kreyssig et al. (2023).

6. A new technique to derive labels for self-supervised learning based on HMMs.

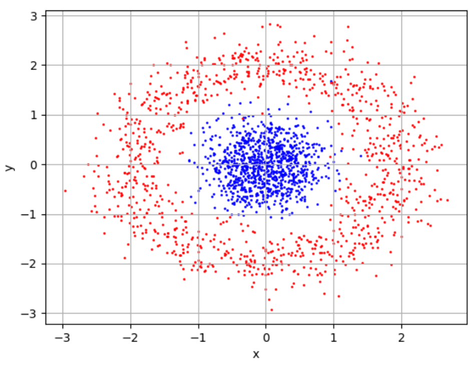
Chapter 2

Deep Learning

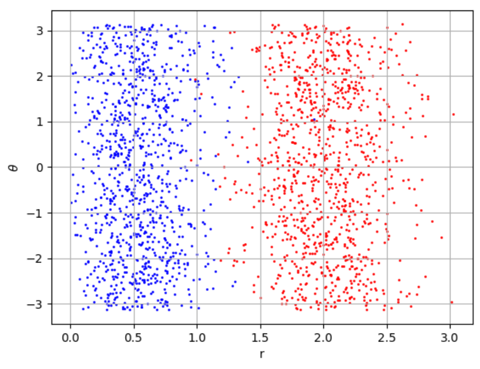
Deep Learning is a subfield of Machine Learning. Deep Learning concerns designing, training and using Artificial Neural Networks (ANNs). Some original neural networks were intended to be computational models of how learning happens or could happen in the brain, which is where the term ANN originates. While parallels can be drawn between ANNs in deep learning and learning in the brain (Hinton and Shallice, 1991), generally ANNs are not, and don't attempt to be, realistic models of the human brain. Deep learning has become the dominant approach for many machine learning tasks such as computer vision (Krizhevsky et al., 2012; Simonyan and Zisserman, 2015), natural language processing (Wang et al., 2019; Wen et al., 2015) and Automatic Speech Recognition (ASR) (Chiu et al., 2018; Dahl et al., 2012; Hinton et al., 2012).

An ANN defines a function $\varphi : \mathcal{X} \rightarrow \mathcal{Y}$ that maps an input $\mathbf{x}$ to an output $\mathbf{y}$. Generally, the ANN will be defined by multiple steps of computation, called *layers*. The input to a specific layer is a vector and the output of this layer is another vector that is passed to other layers (or in the case of a Recurrent Neural Network (RNN) back to the same layer: see Section 2.1.5). There is no agreed-upon definition of a layer, but it generally consists of differentiable operations, usually matrix multiplications, element-wise multiplications, and differentiable functions operating on one or multiple entries of a vector. The common choices for layers are explained in Section 2.1, where they are referred to as the building blocks of an ANN. Due to the ANN being differentiable, its parameters can be optimised, *i.e.* learned, using gradient-based methods, which will be explained in Section 2.2. An entire ANN consists of multiple layers assembling a computational graph with nodes building upon other nodes. Due to the computational graph having many layers of nodes in sequential order, this approach to machine learning is termed *Deep* Learning.

A key part of Deep Learning, and a reason to use many layers, is *representation learning*. The idea is that by using a better representation of the data, the problem can be solved more efficiently. Figure 2.1 illustrates this. A linear classifier, *i.e.* a classifier that can separate



(a) Cartesian



(b) Polar

Figure 2.1 Representation Learning: Cartesian vs Polar representations of the same data set. A linear classifier can separate the two classes when polar coordinates are used, but not when Cartesian coordinates are used.

data into classes using a straight line (or a plane/hyperplane in higher dimensions), cannot separate the blue class from the red class in Figure 2.1a, which uses Cartesian coordinates. However, if a model could learn that using polar coordinates is a better representation of the data (learning this could be the job of the initial layers of a neural network), then the two classes could easily be separated using a linear classifier as seen in Figure 2.1b. Chapter 4 introduces semi-supervised learning where in addition to a labelled data set ($\mathcal{D}_l = \{\mathbf{x}_i, \hat{\mathbf{y}}_i\}_{i=1}^{N}$) an unlabelled data set ($\mathcal{D}_{ul} = \{\mathbf{x}_i\}_{i=N+1}^{N+U}$) exists. Many semi-supervised learning methods attempt to improve the learning of a good representation of the data, after which less labelled data are needed to draw the correct decision boundary.

2.1 Building Blocks

What all building blocks have in common is that they have an input vector $\mathbf{x} = [x_1, \ldots, x_D]^T$ and an output vector $\mathbf{h} = [h_1, \ldots, h_{D'}]^T$. The building blocks define a function $\varphi(\cdot)$ mapping from $\mathbf{x}$ to $\mathbf{h}$ which is parameterised by the block's trainable parameters θ. In many parts throughout this thesis the process of computing $\varphi(\cdot)$ will be referred to as the *forward pass*. To train the ANN using the methods of Section 2.2, the function has to be differentiable, meaning that it must be possible to calculate the partial derivatives $\dfrac{\partial \mathbf{h}}{\partial \mathbf{x}}$ and $\dfrac{\partial \mathbf{h}}{\partial \theta}$.

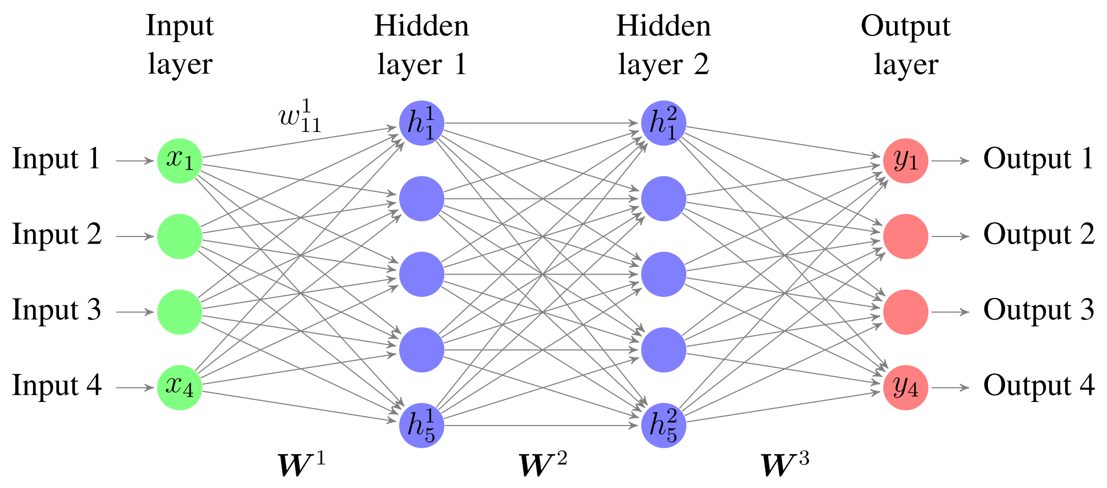


Figure 2.2 A multilayer perceptron with an input size of $D^{(0)} = 4$, an output size of $D^{(3)} = 4$ and two hidden-layers, both of dimensionality $D^{(1)} = D^{(2)} = 5$. Bias vectors are omitted for clarity.

2.1.1 Multilayer Perceptron

The Multilayer Perceptron (MLP) derives its name from the Perceptron formulated by Rosenblatt (1958). The Perceptron is defined as:

$$h = \begin{cases} 1, & \text{if } (\mathbf{x}^T \boldsymbol{w} + b > 0), \\ 0, & \text{otherwise.} \end{cases} \tag{2.1}$$

where the weight vector $\boldsymbol{w} = [w_1, \ldots, w_D]^T$ has the same dimension D as the input vector. The Perceptron was later generalised to what is now known as an artificial neuron:

$$h = \phi\left(\sum_{i=1}^{D} w_i x_i + b\right) = \phi\left(\boldsymbol{w}^T \mathbf{x} + b\right) \tag{2.2}$$

by replacing the Heaviside step function with an arbitrary *activation function* $\phi(\cdot)$ (see Section 2.1.2).

An MLP consists of multiple layers where each layer consists of many artificial neurons. Each layer within the MLP is formed by putting many neurons $h_j^{(l)}$ (defined by Equation (2.2)) into a vector:

$$\mathbf{h}^{(l)} = \phi^{(l)}\left(\boldsymbol{W}^{(l)} \mathbf{x}^{(l)} + \boldsymbol{b}^{(l)}\right) = \phi\left(\mathbf{a}^{(l)}\right) \tag{2.3}$$

where $\mathbf{h}^{(l)}$ is the output of the l^{th} layer and $\mathbf{a}^{(l)}$ are the linear, pre-activation values.

An MLP is an example of a Feedforward Neural Network (FNN), meaning that the output from one layer is passed to succeeding layers only and not any preceding layer or back to itself. In a standard MLP the output from the l-th layer is the input to the l+1-th layer, *i.e.* $\mathbf{y}^{(l)} = \mathbf{x}^{(l+1)}$. If the input feature $\mathbf{x}^{(l)}$ has dimension D_l and the output feature $\mathbf{y}^{(l)}$ has dimension D_{l+1}, then the weight matrix $\boldsymbol{W}^{(l)} \in \mathbb{R}^{D_l \times D_{l+1}}$ and the bias $\boldsymbol{b}^{(l)}$ is a D_{l+1}-dimensional vector.

An MLP is illustrated in Figure 2.2 with $D^{(0)} = 4$, $D^{(1)} = 5 = D^{(2)} = 5$ and $D^{(3)} = 4$. The green nodes represent the input vector $\mathbf{x}$. Each node in blue or red represents an artificial neuron. Each drawn edge is associated with an entry within the weight matrices $\{\boldsymbol{W}^1, \boldsymbol{W}^2, \boldsymbol{W}^3\}$. The bias vector is omitted for clarity. Given that, as drawn, each node of one layer is connected to each node of its succeeding layer, the layers of an MLP (Equation (2.3)) are also called fully-connected (FC) layers. The layers represented in blue, *i.e.* those between the input layer and the output layer ($\mathbf{h}^{(1)}$ and $\mathbf{h}^{(2)}$) are known as *hidden layers* and hence given the letter $\mathbf{h}$. Generally, there are no known "correct" values for the hidden layers $\mathbf{h}^{(l)}$ in comparison to the output layer $\mathbf{h}^{(L)} = \mathbf{y}$. In machine learning in general, hidden variables are also called *latent* variables.

For MLPs, the number of layers (hidden plus output layers) is also known as the depth of the model. The number of neurons of a layer is known as the width of the layer. For an MLP without bias, with a depth of L, where all hidden layers have width $D^{(1)}$, the number of parameters is $D^{(0)}D^{(1)} + (L-2)(D^{(1)})^2 + D^{(1)}D^{(L)}$. Hence, the number of parameters scales linearly with the depth and quadratically with the width.

2.1.2 Activation Functions

Within an FC layer and any other building block of an ANN, the activation function $\phi(\cdot)$ plays a crucial role. Without the activation function, the FC layer, and hence the whole ANN, would be a linear mapping. To make a building block non-linear, non-linear activation functions $\phi(\cdot)$ are used. Generally, activation functions can be separated into those that are used in hidden layers and those used in the output layer, as they have different purposes. Table 2.1 provides a list of widely used activation functions.

Activation functions for hidden layers mainly have the purpose of making the neural network non-linear such that the neural network can represent any function according to the Universal Approximation Theorem (Hornik et al., 1989), assuming sufficient width. Nearly always, for simplicity and computational efficiency, these activation functions operate on each element of the vector individually and are hence called *point-wise* activation functions. Activation functions should be non-linear, continuous and their value as well as their derivative should be cheap to compute.

For the output layer, the activation function should cause the values of the output layer to be structured in such a way that it can be used for a desired task. For a classification task, for example, the output should be a valid probability distribution over the possible classes. The most popular option for this is the Softmax activation function.

$$P(\text{class is } k \mid \mathbf{x}) \;\hat{=}\; y_k = \frac{e^{z_k}}{\sum_{k'=1}^{K} e^{z_{k'}}} = \left[\mathtt{Softmax}\,(\mathbf{z})\right]_k, k = 1, \ldots, K \tag{2.4}$$

where $\mathbf{z}$ is the vector of the pre-activation values $\mathbf{a}^{(L)}$, and is referred to as *logits*. For a regression task, a linear activation function can be used. If the outputs are required to be positive, for example when predicting the variances of a Gaussian Mixture Model (GMM) in a mixture density network (Bishop, 1994), the exponential function can be used to ensure positive outputs.

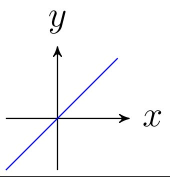
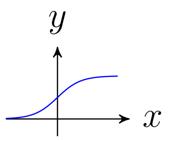
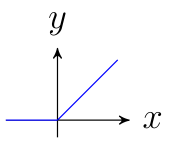
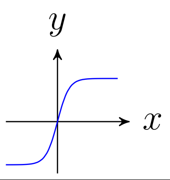
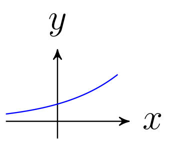
Name	Equation	Derivative	Plot	Purpose
Linear	$\phi(x) = x$	$\phi'(x) = 1$		regression embeddings
sigmoid $\sigma(\cdot)$	$\phi(x) = \frac{1}{1+e^{-x}}$	$\phi'(x) = \phi(x)(1-\phi(x))$		hidden layers binary outputs
ReLU	$\phi(x) = \max(0, x)$	$\phi'(x) = \begin{cases} 0 & , x < 0 \\ 1 & , x \geq 0 \end{cases}$		hidden layers
tanh	$\phi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$\phi'(x) = 1 - \phi(x)^2$		hidden layers
exp	$\phi(x) = e^x$	$\phi'(x) = e^x$		+ve outputs
Softmax	$\phi_i(\mathbf{x}) = \frac{e^{x_i}}{\sum_k e^{x_k}}$	$\frac{\partial \phi_i(\mathbf{x})}{\partial x_j} = \begin{cases} \phi_i(\mathbf{x})(1-\phi_j(\mathbf{x})) & , i = j \\ -\phi_i(\mathbf{x})\phi_j(\mathbf{x}) & , i \neq j \end{cases}$		classification attention

Table 2.1 List of frequently used activation functions, their derivatives and their purposes within neural network structures. Except for the Softmax activation function they operate on each element of a vector individually.

2.1.3 Structured Data

The input vector $\mathbf{x} = [x_1, \ldots, x_D]^T$ was introduced as an unstructured vector without any special relationship between the different indices. That is not the case, however, for many tasks. For example, if the input is an image, then the vector is actually better represented and processed as a 2dim input, *i.e.* instead of using the vector $\mathbf{x} \in \mathbb{R}^{DT}$ the image $\mathbf{X} \in \mathbb{R}^{D\times T}$ is used. In an image, the same important feature (*e.g.* a blob or an edge) to be recognised can occur everywhere in an image. This structure can and should be exploited when designing an ANN. If the input is a sequence, *e.g.* feature vectors derived from audio data, it can be treated as such. Instead of using the vector $\mathbf{x} \in \mathbb{R}^{DT}$ the input is represented as the sequence $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T]$. Time Delay Neural Networks (TDNNs), Convolutional Neural Networks (CNNs) and RNNs are examples of ANN-structures that exploit the structure of the input data. Similarly, the output can have structure. If the output is a sequence, $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_M]$, it could, for example, represent a sentence and each individual output $\mathbf{y}_1$ is a word or a probability distribution over words in a dictionary.

2.1.4 Time-Delay Neural Networks

A TDNN is an extension of the standard MLP such that the input is a sequence $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T]$ and the weights that are applied at different time-steps are the same. A TDNN is composed of FC layers. Each layer in a TDNN takes multiple input vectors at a time as its input and is replicated with at time shifts. At each time shift within the same layer, the exact same FC layer is used (same weight-matrix, same bias-vector). The following layer then takes as input multiple time steps of the preceding layer and is also replicated across different time steps. For input vectors from t-2 to $t{+}2$ for the first layer and also for the second layer this would be:

$$\mathbf{h}_t^{(1)} = \phi^{(1)}\left(\boldsymbol{W}^{(1)}[\mathbf{x}_{t-2}{}^T, \mathbf{x}_{t-1}{}^T, \mathbf{x}_t{}^T, \mathbf{x}_{t+1}{}^T, \mathbf{x}_{t+2}{}^T]^T + \boldsymbol{b}^{(1)}\right) \tag{2.5}$$

$$\mathbf{h}_t^{(2)} = \phi^{(2)}\left(\boldsymbol{W}^{(2)}[\mathbf{h}_{t-2}^{(1)}{}^T, \mathbf{h}_{t-1}^{(1)}{}^T, \mathbf{h}_t^{(1)T}, \mathbf{h}_{t+1}^{(1)}{}^T, \mathbf{h}_{t+2}^{(1)}{}^T]^T + \boldsymbol{b}^{(2)}\right) \tag{2.6}$$

The initial layers thus learn to detect features within narrow temporal contexts, while the later layers operate on a much larger temporal context. Shifting the FC layers across time incorporates the assumption that the same important features can occur at various time steps. Incorporating this knowledge reduces the computation and the number of parameters required. Furthermore, the same FC layer is essentially trained multiple times for the same output label leading to more robust parameters estimation.

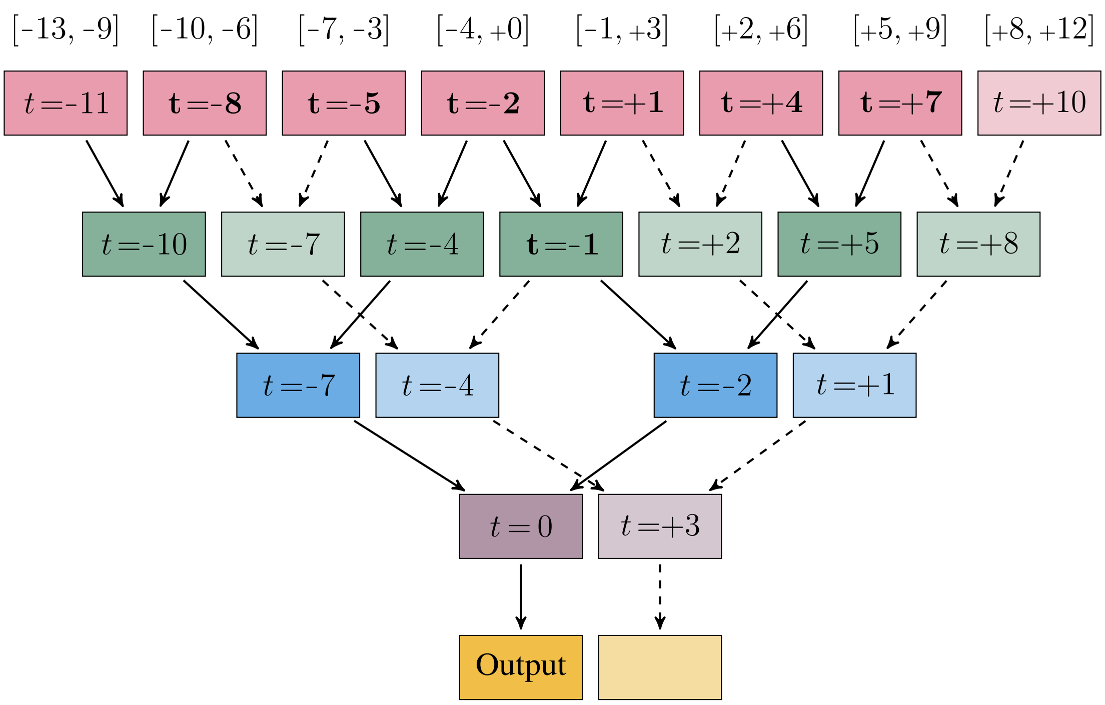


Figure 2.3 Example of a sub-sampled time-delay neural network. Blocks with the same colour share the same parameters. Blocks with bold font are hidden vectors that can be reused when moving the output by 3 time steps. Lighter shaded blocks are hidden vectors that have to be newly calculated when moving the output by 3 time steps. Inputs features are at the top of the figure.

The original TDNN formulation (Waibel, 1989; Waibel et al., 1989) uses shifts of one time-step for the FC layers as in Equation (2.6). This can be very expensive regarding computation and the number of parameters if the input consists of a large number of time-steps. In speech recognition, it is nowadays popular to use time step shifts of more than one within the layers (Peddinti et al., 2015). Non-uniform shifts within the same layer have also been used. Such a *subsampled* TDNN is illustrated in Figure 2.3. It is constructed by moving the first FC layer across the window $t \in [-13, 9]$ with a temporal input each of 5 inputs at shifts of 3 time steps, thus splitting the total input context into 7 time-bins. This is followed by a layer taking in inputs at shifts of -1 and $+2$, followed by a layer taking in inputs at shifts -3 and $+3$, followed by a layer taking in inputs at shifts of -7 and $+2$ followed by the output layer, which produces $\mathbf{y}_t$. The figure also illustrates how much of the computation can be reused when computing the output $\mathbf{y}_{t+3}$ after already having calculated the output $\mathbf{y}_t$. Given that moving the FC Layer across the time axis is akin to the convolution operation in signal processing, each layer is also referred to as a *kernel* as in Figure 2.3. The TDNN moves kernels along one dimension, the time axis, and is a precursor to its multi-dimensional version: the

CNN (LeCun et al., 1989). In most TDNNs the kernel is a FC layer, but multiple layers can also be used (Kreyssig et al., 2018).

2.1.5 Recurrent Neural Networks

RNNs also process sequential data. In an RNN, the computational graph is cyclic, meaning that the output of a layer is fed back into the same layer at the next time step. An RNN can be defined as:

$$(\mathbf{h}_t, \boldsymbol{s}_t) = \text{RNN}\left(\mathbf{x}_t, \boldsymbol{s}_{t-1}\right) \tag{2.7}$$

At time step t, an RNN uses an input $\mathbf{x}_t$ and an internal state $\boldsymbol{s}_{t-1}$ to produce its output $\mathbf{h}_t$ and its new internal state $\boldsymbol{s}_t$. This very general definition allows for many different RNN versions, such as the Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) or the Gated Recurrent Unit (GRU) (Cho et al., 2014). The FC layer of Equation (2.3) can be extended to a "vanilla" RNN (Pineda, 1987; Rumelhart et al., 1988), by concatenating the current input with the previous output of the RNN-layer:

$$\mathbf{h}_t^{(1)} = \phi\left(\boldsymbol{W}^{(1)}\begin{bmatrix}\mathbf{x}_t\\ \mathbf{h}_{t-1}^{(1)}\end{bmatrix} + \boldsymbol{b}^{(1)}\right) \tag{2.8}$$

where $\mathbf{h}_t^{(1)} = \boldsymbol{s}_t^{(1)}$ would be both the output and the internal state. In multi-layered RNNs, the output $\mathbf{h}_t$ is the input to the next recurrent layer at the same time step.

$$\mathbf{h}_t^{(l)} = \phi\left(\boldsymbol{W}^{(l)}\begin{bmatrix}\mathbf{h}_t^{(l-1)}\\ \mathbf{h}_{t-1}^{(l)}\end{bmatrix} + \boldsymbol{b}^{(l)}\right) \tag{2.9}$$

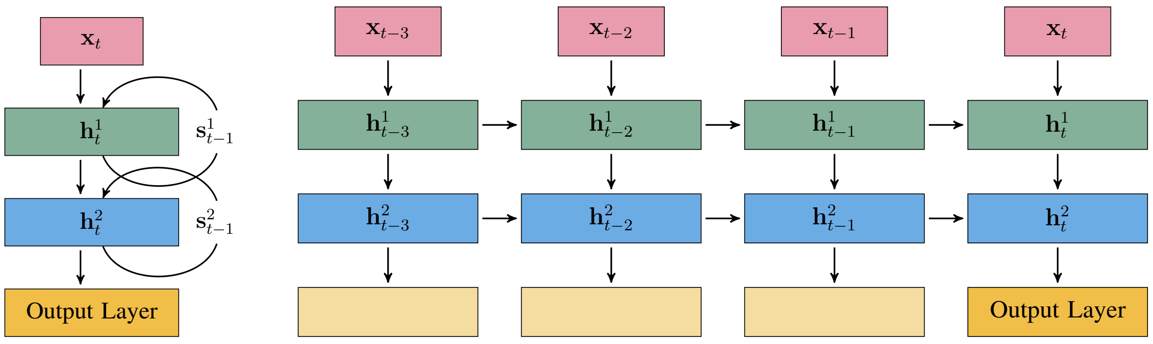
The first layer in such RNN would be defined by Equation (2.8).

Figure 2.4 gives an illustration of a two layer RNN. Both the folded and unfolded representations are given. During training the unfolded representation is used because hidden vectors have to stored for the later gradient calculations. When using a trained RNN, hidden vectors do not have to be stored and hence using the folded representation is more memory efficient.

The RNN of Equation (2.9) is a *uni-directional* RNN, *i.e.* the current output only depends on the past. In a *bi-directional* RNN (Schuster and Paliwal, 1997) the first layer consists of one RNN, $\overrightarrow{\mathbf{h}_t^{(1)}}$, that only depends on the past and one RNN, $\overleftarrow{\mathbf{h}_t^{(1)}}$, that only depends on the future.

$$\overrightarrow{\mathbf{h}_t^{(1)}} = \phi\left(\overrightarrow{\boldsymbol{W}^{(1)}}\begin{bmatrix}\mathbf{x}_t\\ \overrightarrow{\mathbf{h}_{t-1}^{(1)}}\end{bmatrix} + \overrightarrow{\boldsymbol{b}^{(1)}}\right) \qquad \overleftarrow{\mathbf{h}_t^{(1)}} = \phi\left(\overleftarrow{\boldsymbol{W}^{(1)}}\begin{bmatrix}\mathbf{x}_t\\ \overleftarrow{\mathbf{h}_{t+1}^{(1)}}\end{bmatrix} + \overleftarrow{\boldsymbol{b}^{(1)}}\right) \tag{2.10}$$

where $\overrightarrow{\boldsymbol{W}}^{(1)}$ and $\overleftarrow{\boldsymbol{W}}^{(1)}$ are separate weight matrices for the forward and the backward RNN. The next layer (either another RNN-layer or an output layer) then takes the concatenation of $\overrightarrow{\mathbf{h}}_t^{(1)}$ and $\overleftarrow{\mathbf{h}}_t^{(1)}$ as its input $\mathbf{x}_t^{(2)} = [\overrightarrow{\mathbf{h}}_t^{(1)\,T}, \overleftarrow{\mathbf{h}}_t^{(1)T}]^T$.



(a) Folded (b) Unfolded

Figure 2.4 A two layer Recurrent Neural Network (RNN) in both its folded and unfolded representation. Blocks represent fully-connected layers. Blocks that have the same colour have the same parameters.

2.1.5.1 Long Short-Term Memory

Since the output of an RNN layer is passed back to its input at the next time step, an RNN can be seen as a potentially infinitely deep ANN. This can cause a problem called *vanishing* or *exploding* gradients (Bengio et al., 1994), which is explained in more detail in Section 2.2. When the sigmoid activation function is used, it's mainly vanishing gradients that occur. This problem was addressed by Hochreiter and Schmidhuber (1997) by introducing the LSTM RNN, defined by:

$$\tilde{\boldsymbol{c}}_t = \mathtt{tanh}\left(\boldsymbol{W}_c\mathbf{x}_t + \boldsymbol{V}_c\mathbf{h}_{t-1} + \boldsymbol{b}_c\right) \tag{2.11}$$

$$\boldsymbol{i}_t = \sigma\left(\boldsymbol{W}_i\mathbf{x}_t + \boldsymbol{V}_i\mathbf{h}_{t-1} + \boldsymbol{b}_i\right) \tag{2.12}$$

$$\boldsymbol{f}_t = \sigma\left(\boldsymbol{W}_f\mathbf{x}_t + \boldsymbol{V}_f\mathbf{h}_{t-1} + \boldsymbol{b}_f\right) \tag{2.13}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t \tag{2.14}$$

$$\boldsymbol{o}_t = \sigma\left(\boldsymbol{W}_o\mathbf{x}_t + \boldsymbol{V}_o\mathbf{h}_{t-1} + \boldsymbol{b}_o\right) \tag{2.15}$$

$$\mathbf{h}_t = \boldsymbol{o}_t \odot \mathtt{tanh}(\boldsymbol{c}_t) \tag{2.16}$$

The input to the LSTM, is the input vector at time t $\mathbf{x}_t$, the output vector of the previous time step $\mathbf{h}_{t-1}$ and the memory cell vector of the previous time step $\boldsymbol{c}_{t-1}$. Inside the LSTM, three gates modulate the flow of information: the input gate $\boldsymbol{i}_t$, the forget gate $\boldsymbol{f}_t$ and the output gate $\boldsymbol{o}_t$. Matrices $\boldsymbol{W}$ and $\boldsymbol{V}$ are the weight matrices associated with each gate or with the proposed memory cell $\tilde{\boldsymbol{c}}_t$. $\sigma(\cdot)$ is the sigmoid activation function and $\odot$ denotes element-wise multiplication. The element-wise multiplications ($\odot$) and especially the addition operation in Equation (2.14) help to mitigate the problem of vanishing gradients. Here, the state of Equation (2.7) is given by $\boldsymbol{s}_t = \{\mathbf{h}_t, \boldsymbol{c}_t\}$.

2.1.6 Attention Mechanism

The RNNs of Section 2.1.5 can summarise any sequence of vectors $\mathbf{x}_{1:T}$ into a state $\boldsymbol{s}_T$, by recursively applying the same function (Equation (2.7)). Another method is to calculate a weighted average of the sequence:

$$\mathbf{h} = \sum_{t=1}^{T} \alpha_t \mathbf{x}_t \tag{2.17}$$

where α_t is a set of weights. In this case, $\mathbf{x}_{1:T}$ can be either any sequence of input or hidden vectors. The context vector $\mathbf{h}$ is generally passed through further layers to generate the output vector $\mathbf{y}$. In an *attention mechanisms*, the weights $\alpha_{1:T}$ are computed dynamically *i.e.* they depend on the input data itself. In this way, the ANN can selectively pay attention to different parts of the sequence $\mathbf{x}_{1:T}$ when needed. In most cases $\alpha_{1:T}$ will depend not only on $\mathbf{x}_{1:T}$ but also on some additional input $\boldsymbol{k}$. There are multiple ways of using $\mathbf{x}_{1:T}$ and $\boldsymbol{k}$ to produce $\alpha_{1:T}$. One way to compute $\alpha_{1:T}$ is through another ANN $\varphi(\cdot)$ (*e.g.* an MLP in (Bahdanau et al., 2014)) followed by the Softmax activation function:

$$e_t = \varphi\left(\mathbf{x}_t, \mathbf{k}\right) \tag{2.18}$$

$$\alpha_t = \frac{e^{e_t}}{\sum_{j=1}^{T} e^{e_j}}, t = 1, \dots, T \tag{2.19}$$

This results in values for $\alpha_{1:T}$ that are positive and sum to one.

Attention mechanisms are useful for any task in which it is unknown which portion of the sequence $\mathbf{x}_{1:T}$ is informative for further computations, but at the same time concatenating $\mathbf{x}_{1:T}$ to a large vector and feeding it into further layers would lead to using too many parameters. Instead of a sequence, it is equivalently possible to attend over a multi-dimensional input such as an image.

Attention mechanisms are often used in tasks that output a sequence. This includes handwriting generation (Graves, 2013), machine translation (Bahdanau et al., 2014), image captioning (Xu et al., 2015) and ASR (Chan et al., 2016). The input and output sequence will generally not have the same length. At each time step m of the output sequence, the attention mechanism will compute weights $\alpha_{m,t}$ and thus a new hidden vector $\mathbf{h}_m$. The attention mechanism will need to learn which part of the input sequence is important for the generation of the output $\mathbf{y}_m$. For this, a representation of the previous outputs $\mathbf{y}_{1:m\text{-}1}$ as $\mathbf{k}_m$ can be useful.

2.1.7 Self- and Source-Attention

The Transformer (Vaswani et al., 2017) is a widely used neural network architecture for sequence-to-sequence modelling and does not use any recurrence. It relies solely on the attention mechanism and has outperformed RNN-based models in many language-related tasks (Devlin et al., 2019). The Transformer's key concept is the scaled dot-product attention mechanism. The input to this particular attention mechanism consists of queries $\boldsymbol{E}_q \in \mathbb{R}^{D_k \times T_q}$, keys $\boldsymbol{E}_k \in \mathbb{R}^{D_k \times T_k}$ and values $\boldsymbol{E}_v \in \mathbb{R}^{D_v \times T_k}$. They are obtained by the following linear transforms:

$$\begin{aligned}
\boldsymbol{E}_q &= \boldsymbol{W}_q \mathbf{X}_q, &\quad \text{where } \boldsymbol{W}_q \in \mathbb{R}^{D_k \times D_{X_q}}, \mathbf{X}_q \in \mathbb{R}^{D_{X_q} \times T_q} \\
\boldsymbol{E}_k &= \boldsymbol{W}_k \mathbf{X}_k, &\quad \text{where } \boldsymbol{W}_k \in \mathbb{R}^{D_k \times D_{X_k}}, \mathbf{X}_q \in \mathbb{R}^{D_{X_k} \times T_k} \\
\boldsymbol{E}_v &= \boldsymbol{W}_v \mathbf{X}_v, &\quad \text{where } \boldsymbol{W}_v \in \mathbb{R}^{D_v \times D_{X_v}}, \mathbf{X}_q \in \mathbb{R}^{D_{X_v} \times T_k}
\end{aligned} \tag{2.20}$$

The input sequence for the values $\mathbf{X}_v$ can be viewed as the actual input sequence that is being mapped and it is of the same length as the input sequence of the keys $\mathbf{X}_k$. The input sequence for the queries $\mathbf{X}_q$ will be of the same length as the output sequence.

In order to calculate the attention weights, the attention mechanism first calculates the similarity between the query and key vectors. This is done by taking the dot product of the query and key vectors and scaling it down by $\sqrt{D_k}$, where D_k is the dimension of the key vectors. The dot product values represent the closeness between the query and key vectors, with higher values indicating a closer match. Next, the dot product values are transformed into a valid probability distribution with the Softmax function. These normalised values are used as weights for the weighted sum of the value vectors, which results in one feature vector for each time step in the query.

$$\textsc{Attention}(\boldsymbol{E}_q, \boldsymbol{E}_k, \boldsymbol{E}_v) = \boldsymbol{E}_v \ \texttt{Softmax}\left(\frac{\boldsymbol{E}_k^T \boldsymbol{E}_q}{\sqrt{D_k}}\right) \tag{2.21}$$

where the Softmax is applied to each column and the result of the attention operation has dimension $T_v \times D_q$.

In the Transformer (Vaswani et al., 2017), two forms of this attention are used. The first is self-attention where $\mathbf{X}_q = \mathbf{X}_k = \mathbf{X}_v$. This is used to encode sequences and the length of the input sequence and the output sequence is the same. The second form of attention is source-attention, where $\mathbf{X}_k = \mathbf{X}_v$. Here, $\mathbf{X}_k$ is usually the encoding provided by a self-attention model. The output sequence will be of the same length as $\mathbf{X}_q$, which is often provided by an encoding of the history sequence.

Transformer models are generally built through multiple encoder and decoder blocks. An encoder block consists of one self-attention mechanism, followed by a two-layer MLP that is applied to each output vector individually. Both the self-attention operation and the MLP use residual connections (He et al., 2016) which are followed by layer normalisation (Ba et al., 2016). A decoder block consists of one self-attention mechanism, followed by one source-attention mechanism, which is followed by a two-layer MLP. As before, all three employ residual connections that are followed by layer normalisation.

Often, multi-head attention is used. In multi-head attention, first, each vector in each $\boldsymbol{E}$ is split into N "heads". This effectively leads to N different sequences being considered. The attention mechanism is applied to each head N individually and the output vectors are concatenated. This is followed by a linear mapping using another weight matrix $\boldsymbol{W}_o \in \mathbb{R}^{D_v \times D_v}$. The effect of this is that a different weighted sum can be used for each head. In turn, this means that each head can selectively pay attention (*i.e.* focus) on different points in time.

The Transformer model is permutation equivariant and hence does not consider the ordering of the input sequence. This is addressed through positional encoding. In the original Transformer model (Vaswani et al., 2017), the positional encoding is additive based on sinusoidal functions. Another option that is popular in ASR is relative positional encoding using convolutional neural networks (Mohamed et al., 2019) which is used in the experiments in Chapter 9.

2.1.8 Emformer

The Transformer architecture of Section 2.1.7 is very effective at modelling long sequences of data. Rather than using memory state to capture long-range dependencies in recurrent neural networks, the multi-head self-attention method connects arbitrary positions in the whole sequence directly in parallel. In comparison to the natural language processing tasks that initially popularised the Transformer architectures, many ASR applications require streaming ASR. Whilst many approachest have been developed for making the Transformer architecture streaming for ASR (Dong et al., 2019; Li et al., 2021b; Moritz et al., 2020; Povey et al., 2018; Wu et al., 2020; Zhang et al., 2020), this section focuses on the Emformer architecture (Shi et al., 2021). This particular model architecture is not only streaming, *i.e.* it can operate under strict latency constraints, but it is also RAM limited and can hence be used on mobile devices. It will be used for the experiments in Chapter 9 and some issues relating to it motivate the work in Section 9.4.

The Emformer uses a block processing method to make the task streaming. Within each block, the Emformer performs in a similar way to the Transformer. The blocks are overlapping and consist of a right-context and a main segment. Shi et al. (2021) also uses a memory-bank and a left-context. Recent publications that use the Emformer architecture and this thesis do not use them.

The input to an Emformer-layer is “chopped” into non-overlapping blocks that have the same size as the main segment. Each block is appended to the right (*i.e.* into the future) by the right-context, which leads to each block overlapping slightly. Given that the Transformer produces all outputs at the same time. The average algorithmic latency of an Emformer is the right-context plus half the size of the main segment.

2.2 Parameter Optimisation

For any ANN the goal of learning is to determine suitable parameters θ (*i.e.* weight matrices and bias vectors). This is done by first choosing a suitable cost function $\mathcal{C}(\theta)$ and altering θ to reduce $\mathcal{C}(\theta)$. The simplest algorithm to optimise $\mathcal{C}(\theta)$ in ANNs, is called *Gradient Descent*. It involves finding the gradient of $\mathcal{C}(\theta)$ w.r.t. θ, $\frac{\partial \mathcal{C}}{\partial \theta}$, and then changing the weights in the opposite direction to the gradient:

$$\theta^{(u)} = \theta^{(u-1)} - \eta \left.\frac{\partial \mathcal{C}}{\partial \theta}\right|_{\theta^{(u-1)}} \tag{2.22}$$

$$\Delta\theta^{(u)} = -\eta \left.\frac{\partial \mathcal{C}}{\partial \theta}\right|_{\theta^{(u-1)}} \tag{2.23}$$

where η is called the *learning rate* or *step-size* and u is the current update iteration. The cost $\mathcal{C}(\theta)$ is generally defined over a labelled training set ($\mathcal{D}_l = \{\mathbf{x}_i, \hat{\mathbf{y}}_i\}_{i=1}^{N}$), usually in a way that assumes independence between the data samples:

$$\mathcal{C}(\theta, \mathcal{D}_l) = \sum_{i=1}^{N} \mathcal{L}(\theta, \mathbf{x}_i, \hat{\mathbf{y}}_i) \tag{2.24}$$

$\mathcal{L}(\theta, \mathbf{x}_i, \hat{\mathbf{y}}_i)$ is called the *loss-function*.

2.2.1 Loss Functions

The loss function $\mathcal{L}(\theta, \mathbf{x}_i, \hat{\mathbf{y}}_i)$ is usually computed by first computing the output of the model $\mathbf{y}_i$ from $\mathbf{x}_i$ based on the model's parameters θ and then comparing it to the “correct” output $\hat{\mathbf{y}}_i$. For a regression task, $\hat{\mathbf{y}}_i$ would be a real-valued vector. Here, a popular loss function is the squared error,

$$\mathcal{L}(\theta, \mathbf{x}_i, \hat{\mathbf{y}}_i) = \|(\mathbf{y}_i - \hat{\mathbf{y}}_i)\|^2 \tag{2.25}$$

which turns the cost $\mathcal{C}$ into the Mean Squared Error (MSE). For a classification task, $\hat{\mathbf{y}}_i$ would be a one-hot vector to represent the correct class or a categorical distribution that represents "soft" labels. As mentioned before, in classification tasks the Softmax activation function (Equation (2.4)) is usually used, and its output is interpreted as a probability distribution over the possible classes that the ANN assigns to the input $\mathbf{x}$. An intuitive objective is to train the ANN such that it assigns the highest probability possible to the N training samples $\mathcal{D}_l = \{\mathbf{x}_i, \hat{\mathbf{y}}_i\}_{i=1}^N$.

$$\theta^* = \underset{\theta}{\text{argmax}} \prod_{i=1}^{N} P_\theta(\hat{\mathbf{y}}_i \mid \mathbf{x}_i) = \underset{\theta}{\text{argmin}} -\sum_{i=1}^{N} \log P_\theta(\hat{\mathbf{y}}_i \mid \mathbf{x}_i) \tag{2.26}$$

This is called Maximum Likelihood (ML) training or Conditional Maximum Likelihood (CML) given that it's conditional on input $\mathbf{x}_i$.

$$\begin{aligned} \theta^* &= \underset{\theta}{\text{argmin}} -\sum_{i=1}^{N}\sum_{k=1}^{K} \hat{y}_{ik} \log P_\theta(\hat{\mathbf{y}}_i \mid \mathbf{x}_i) && (2.27)\\ &= \underset{\theta}{\text{argmin}} -\sum_{i=1}^{N}\sum_{k=1}^{K} \hat{y}_{ik} \log y_{ik} && (2.28)\\ \therefore \mathcal{C}(\theta) &= -\sum_{i=1}^{N}\sum_{k=1}^{K} \hat{y}_{ik} \log y_{ik} && (2.29)\\ \therefore \mathcal{L}(\theta, \mathbf{x}_i, \hat{\mathbf{y}}_i) &= -\sum_{k=1}^{K} \hat{y}_{ik} \log y_{ik} && (2.30) \end{aligned}$$

This loss function is widely used in classification and usually referred to by its name from Information Theory called Cross Entropy (CE), though it is more intuitive to derive it through the ML framework.

In the next section the gradients of $\mathcal{L}(\theta, \mathbf{x}_i, \hat{\mathbf{y}}_i)$ w.r.t. to the parameters θ will be considered. To obtain the gradients for $\mathcal{C}(\theta)$ these equations need to be summed over all samples. Loss functions are also often chosen so that their gradients are easy to compute. Replacing the squaring operation in Equation (2.25) with an absolute value would lead to the loss function having a discontinuity, which is one of the reasons why MSE is more popular. If the Softmax activation function is used at the output, the gradient of the loss-function from Equation (2.30) w.r.t. the logits ($\mathbf{z}$ in Equation (2.4)), has the simple closed form expression given by:

$$\frac{\partial \mathcal{L}}{\partial z_{ik}} = \hat{y}_{ik} - y_{ik} \tag{2.31}$$

This is also the gradient of the MSE loss if a linear output activation function is used.

2.2.2 Backpropagation

For an FC layer, given the gradient of the cost $\mathcal{C}$ w.r.t. the linear activations $\mathbf{a}^{(l)}$ of hidden layer l $\frac{\partial \mathcal{C}}{\partial \mathbf{a}^{(l)}}$, finding the gradients w.r.t. the parameters of the hidden layer is trivial. These gradients are given by:

$$\frac{\partial \mathcal{C}}{\boldsymbol{W}_{ij}^{(l)}} = \frac{\partial \mathcal{C}}{\partial a_i^{(l)}} \, h_j^{(l-1)} \quad \longrightarrow \quad \frac{\partial \mathcal{C}}{\partial \boldsymbol{W}^{(l)}} = \frac{\partial \mathcal{C}}{\partial \mathbf{a}^{(l)}} \, \mathbf{h}^{(l-1)T} \tag{2.32}$$

$$\frac{\partial \mathcal{C}}{\partial b_i} = \frac{\partial \mathcal{C}}{\partial a_i^{(l)}} \quad \longrightarrow \quad \frac{\partial \mathcal{C}}{\partial \boldsymbol{b}^{(l)}} = \frac{\partial \mathcal{C}}{\partial \mathbf{a}^{(l)}} \tag{2.33}$$

The challenge comes when trying to use $\frac{\partial \mathcal{C}}{\partial \mathbf{a}^{(l)}}$ to find $\frac{\partial \mathcal{C}}{\partial \mathbf{a}^{(l-1)}}$. This would be to *propagate* the gradients *back* from one layer to an earlier layer, which is where the term *back-propagation* originates. This can be achieved by applying the chain rule of calculus:

$$\frac{\partial \mathcal{C}}{\partial \mathbf{a}^{(l-1)}} = \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{a}^{(l-1)}} \frac{\partial \mathcal{C}}{\partial \mathbf{a}^{(l)}} \tag{2.34}$$

$$= \frac{\partial \mathbf{h}^{(l-1)}}{\partial \mathbf{a}^{(l-1)}} \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{h}^{(l-1)}} \frac{\partial \mathcal{C}}{\partial \mathbf{a}^{(l)}} \tag{2.35}$$

The gradient of the linear mapping is given by,

$$\frac{\partial a_i^{(l)}}{\partial h_j^{(l-1)}} = \left[\boldsymbol{W}^{(l)}\right]_{ji} = \left[\boldsymbol{W}^{(l)T}\right]_{ij} \tag{2.36}$$

which leads to the overall gradient being given by,

$$\frac{\partial \mathcal{C}}{\partial \mathbf{a}^{(l-1)}} = \frac{\partial \mathbf{h}^{(l-1)}}{\partial \mathbf{a}^{(l-1)}} \boldsymbol{W}^{(l)T} \frac{\partial \mathcal{C}}{\partial \mathbf{a}^{(l)}} \tag{2.37}$$

This process can be initialised using the gradients from Equation (2.31), because for the output layer $\mathbf{a}^{(L)} = \mathbf{z}$. Note that, $\frac{\partial \mathbf{h}^{(l-1)}}{\partial \mathbf{a}^{(l-1)}}$ is effectively the gradient of the activation function $\phi^{(l-1)}(\cdot)$. As shown in Table 2.1, most activation functions are element-wise, and thus $\frac{\partial \mathbf{h}^{(l-1)}}{\partial \mathbf{a}^{(l-1)}}$ is a diagonal matrix, with its elements along the diagonal given by the equations given in Table 2.1. The whole process of calculating all the gradients for the activations and the weights of an ANN or a building block is referred to as the *backwards pass* because it is the counterpart to the forward pass.

From Equation (2.37) it can be seen that if the spectral radius of $\frac{\partial \mathbf{h}^{l-1}}{\partial \mathbf{a}^{l-1}} \boldsymbol{W}^{(l)T}$ is above 1 for each layer, then the gradients will become ever larger as the gradients are propagated back to earlier layers. If the spectral radius is below one then the gradients will become ever smaller. These two problems are the *exploding* and *vanishing* gradients mentioned in Section 2.1.5. The gradient of the sigmoid activation function is always below 0.25. Furthermore, regularisation techniques are sometimes used that try to limit the sizes of the individual weights within a weight matrix, such as weight-decay (Section 2.3). Such a technique leads to a smaller spectral radius of $\boldsymbol{W}^{(l)T}$. Therefore, *vanishing* gradients are more often a problem within deep learning. Due to the depth of RNNs being potentially infinite, such issues occur more often in RNNs. Given that the derivative of the ReLU activation function is either 0 or 1, ANNs that use the ReLU activation function often don't suffer as much from vanishing gradients as those with sigmoid activation functions.

2.2.3 Stochastic Gradient Descent

Equation (2.22) calculates gradients of the cost based on the entire data set. This is a computationally expensive process for large data sets. However, in most cases the gradient can be reasonably well approximated based on only a small subset (a so-called *minibatch)* of the data set:

$$\mathcal{C}^{\text{MB}}(\theta) = -\sum_{i=1}^{B} \sum_{k=1}^{K} y_{ik} \log \hat{y}_{ik} \tag{2.38}$$

where B is the minibatch size with usually $B \ll N$. When using such a minibatch the learning rate η obviously needs to be scaled up appropriately. For each update, a different minibatch is sampled at random without replacement from the training set until it is exhausted. Training on all samples of the training set once is referred to as one *epoch*. The entire procedure is then referred to as Stochastic Gradient Descend (SGD). SGD leads to a drastic reduction in computation cost per update. The minibatch size can be set such that the computational power available allows the entire batch to be calculated in parallel. At the same time, however, it also means that the individual gradients are a noisy estimate of the true gradient. This can lead to a lower convergence speed in terms of the number of updates needed to reach an optimum. At the same time, however, SGD allows for the possibility of escaping a local optimum since the optimum w.r.t. the error surface for of the entire data set is likely not an optimum for a specific minibatch (Bishop, 2006). Many other popular optimisers exist such as Adam (Kingma and Ba, 2015), AdaGrad (Duchi et al., 2011) or RMSProp (Tieleman and Hinton, 2012).

2.2.3.1 Learning Rate Scheduling

Optimisation using gradient descent can have issues related to the learning rate η. First, if η is set too large, training might never converge and might even diverge. For example, taking for the simple function (not a neural network):

$$f(x) = x^2; \quad \frac{\partial f}{\partial x} = 2x \tag{2.39}$$

For $f(x_1 = 1) = 1$ the gradient is $\frac{\partial f}{\partial x} = 2x_1$. Hence, for $\eta = 1.5$ the updated value for x would be $x_2 = 1 - 1.5 \cdot 2 \cdot x_1 = -2$ and then $x_3 = 2 - 1.5 \cdot 2 \cdot x_2 = 4$ and so on. This simple example shows how a too high learning rate η leads to divergence. In general, within an ANN a too high learning rate will lead to more noisy updates and large oscillations of the loss function. Even if divergence does not occur, it can still lead to much slower convergence. A too low learning rate will cause slower learning and also might cause the network to be stuck in a bad local optimum with a high cost value. A larger learning rate might be able to "jump" out of this local optimum. Therefore the learning rate η has to be carefully picked.

Approximating the training cost function for $\Delta\theta = -\eta \frac{\partial \mathcal{C}}{\partial \theta}\Big|_{\theta^{(u-1)}} = -\eta\mathbf{g}$ using the truncated Taylor series leads to:

$$\mathcal{C}(\theta + \Delta\theta) = \mathcal{C}(\theta) - \eta\mathbf{g}^T\mathbf{g} + \tfrac{1}{2}\eta^2\mathbf{g}^T\mathbf{H}\mathbf{g} \tag{2.40}$$

As long as the term $\frac{1}{2}\eta^2\mathbf{g}^T\mathbf{H}\mathbf{g}$ is small, gradient descent will decrease $\mathcal{C}$. However, it turns out that as training proceeds it is very common for the term $\mathbf{g}^T\mathbf{H}\mathbf{g}$ to grow significantly (Goodfellow et al., 2016, Chapter 8.2). Hence, as training proceeds the learning rate η needs to be reduced in order to keep the second-order term small. One very common technique is to decay the learning rate exponentially:

$$\eta_t = \eta_0 e^{-\frac{u}{\tau}} \tag{2.41}$$

where τ is the decay constant and η_0 is the initial learning rate. A technique such as Equation (2.41) is known as a learning rate scheduler. Another learning rate scheduler used throughout this thesis is the NewBob learning rate scheduler. The NewBob learning rate scheduler also decays the learning rate exponentially. However, instead of decaying based on the number of updates/epochs, the learning rate is decayed based on the performance of the ANN. Whenever the cost on the validation set stops decreasing the learning rate is decayed by a certain factor. The NewBob scheduler was used for the experiments in Chapters 6, 7 and 8. Further refinements can be applied to the gradient to improve training stability. Derivatives in ANNs can sometimes have a very large magnitude. In these cases, it can be useful to "clip" the gradients. This can be done by setting a maximum absolute value for the elements of the gradient vector

(Mikolov, 2012) or by setting a maximum value for the norm of the gradient vector (Pascanu et al., 2013). The former was used often in this thesis.

2.2.3.2 Momentum

The momentum method, originally proposed by Polyak (1964), is an extension of SGD that can lead to significantly faster convergence for many optimisation tasks including training ANNs. When using momentum instead of using the gradient in Equation (2.22), an exponentially smoothed version of the gradient, the momentum vector $\boldsymbol{v}$, is used:

$$\boldsymbol{v}^{(u)} = \omega \boldsymbol{v}^{(u-1)} + (-\eta \left. \frac{\partial \mathcal{C}}{\partial \theta} \right|_{\theta^{(u-1)}}) \tag{2.42}$$

$$\Delta\theta^{(u)} = \boldsymbol{v}^{(u)} \tag{2.43}$$

$$\theta^{(u)} = \theta^{(u-1)} + \Delta\theta^{(u)} \tag{2.44}$$

where $\omega \in [0, 1]$ is the momentum parameter and η is the learning rate as before. Directions of low curvature will persist over time and hence they will be amplified. Second-order optimisation methods amplify directions of low curvature by scaling each eigen-direction of the curvature matrix by the inverse of the associated curvature (eigenvalue) (Sutskever et al., 2013). Therefore, momentum is sometimes seen as a “pseudo”-second-order method. The popular optimiser Adam (Kingma and Ba, 2015) extends the momentum method by not using one global momentum parameter ω, but by associating individual momentum parameters with each trainable parameter.

2.2.4 Initialisation

Before the parameters can be updated for the first time (*e.g.* using Equation (2.22) or (2.43)), an initial value needs to be picked for $\theta^{(0)}$. The initial value for $\theta^{(0)}$ can have a significant impact on the learning procedure and the final performance of the model (Sutskever et al., 2013). Too large parameter values can cause all sigmoid or $\tanh$ units to saturate and therefore in turn cause the gradients to vanish. Similarly, if the weights are too small the gradients will vanish as they are back-propagated through the weight matrices. The Glorot initialisation method (Glorot and Bengio, 2010) is designed for fully-connected layers (this also includes the “fully-connected” layers that build the LSTM layer, *e.g.* Equation (2.11)). The weights

$[\boldsymbol{W}]_{ij}^{(l)} = w_{ij}^{(l)}$ are sampled i.i.d. from one of the following distributions:

$$w_{ij}^{(l)} \sim \mathcal{U}(-u, u) \text{ where } u = \gamma\sqrt{\frac{3}{\frac{1}{2}(D^{(l)} + D^{(l-1)})}}, \quad \text{or} \tag{2.45}$$

$$w_{ij}^{(l)} \sim \mathcal{N}(0, \sigma^2) \text{ where } \sigma = \gamma\sqrt{\frac{1}{\frac{1}{2}(D^{(l)} + D^{(l-1)})}}, \tag{2.46}$$

where $\mathcal{U}$ and $\mathcal{N}$ are uniform and normal distributions. Glorot and Bengio (2010) only considered the linear activation function. The gain factor γ was introduced by He et al. (2015) to compensate for the scaling of the variance due to the activation function *e.g.* for an input with a symmetric distribution centred on zero, the ReLU activation function maps half of the activations to zero, in turn scaling down the variance. Therefore, the gain factor for the ReLU activation function is above 1 (specifically $\gamma = \sqrt{2}$). For the sigmoid activation function and the linear activation function $\gamma = 1$ and for tanh $\gamma = 5/3$. The denominator can be replaced by only $D^{(l)}$ to preserve the variance during the backwards pass or by $D^{(l-1)}$ to preserve the variance during the forward pass.

2.3 Regularisation

For many machine learning methods, it is common for the loss on the training set to be lower than on a held-out test set. This is especially true for deep learning where the number of parameters can be very large and in some cases much larger than the number of data points in the training set. This phenomenon is called *overfitting*. In ANNs overfitting generally results in the model memorising the data rather than finding patterns in the training data that are relevant to unseen data. *Regularisation* refers to methods that try to prevent the model from overfitting. They generally make it harder for the model to optimise the training loss, but ideally in such a way that the loss on a held-out test set goes down. The difference in performance between the training set and test set is generally referred to as the *generalisation error*. Therefore, regularisation can be described as methods that try to reduce the generalisation error such that the performance on the test set improves.

2.3.1 Weight-Decay

In Bayesian approaches to machine learning, parameters are treated as random variables. Instead of finding the parameters that assign the highest probability to the training data (*i.e.* $\operatorname*{argmax}_{\theta}\, p_{\theta}\left(\{\mathbf{x}_i, \hat{\mathbf{y}}_i\}_{i=1}^{N}\right)$ as in ML Training), the parameters with the highest probability given the training data are chosen (*i.e.* $\underset{\theta}{\operatorname{argmax}}\, p\left(\theta | \left\{\mathbf{x}_i, \hat{\mathbf{y}}_i\right\}_{i=1}^N\right)$). This is referred to as Maximum a Posteriori (MAP) Training. Based on Bayes' rule, MAP training is equivalent to optimising $\underset{\theta}{\operatorname{argmax}} \left[p_\theta\left(\left\{\mathbf{x}_i, \hat{\mathbf{y}}_i\right\}_{i=1}^N\right) p(\theta)\right]$, where $p(\theta)$ is a *prior* over the parameters. A popular prior in many areas of Bayesian machine learning is an independent Gaussian prior over all parameters $p(\theta) = \mathcal{N}\left(\theta; \mathbf{0}, \frac{1}{\lambda}\mathbf{I}\right)$ (Hernandez-Lobato and Adams, 2015). Based on this prior, the cost function from Equation (2.30) can be modified to:

$$\mathcal{C}(\theta) = -\log\left(p_\theta\left(\left\{\mathbf{x}_i, \hat{\mathbf{y}}_i\right\}_{i=1}^N\right) p(\theta)\right) \tag{2.47}$$

$$\mathcal{C}(\theta) = -\log\left(p(\theta)\right) - \sum_{i=1}^{N} \log P_\theta\left(\hat{\mathbf{y}}_i | \mathbf{x}_i\right) + \text{const. terms} \tag{2.48}$$

$$\mathcal{C}(\theta) = \frac{\lambda}{2}\left\|\theta\right\|^2 - \sum_{i=1}^{N}\sum_{k=1}^{K} \hat{y}_k^i \log y_k^i \qquad + \text{const. terms} \tag{2.49}$$

The reciprocal of the prior variance λ, is referred to as the weight-decay parameter. This is also equivalent to using the ML objective (Equation (2.29)) and adding a term to the update equation (Equation (2.22)):

$$\theta_{\text{new}} = \theta_{\text{old}} \cdot (1 - \eta\lambda) - \eta \left.\frac{\partial \mathcal{C}}{\partial \theta}\right|_{\theta_{\text{old}}} \tag{2.50}$$

Thus, the use of the Gaussian prior onto the weights is equivalent to shrinking the weights by a constant factor ($(1 - \eta\lambda)$) at each iteration. Due to this interpretation, this regulariser is usually referred to as *weight-decay*. The Bayesian interpretation is however useful when deciding the value of the weight-decay parameter λ. Based on Equation (2.49) if SGD is used, it is clear that the weight-decay parameter λ should be scaled in inverse proportion with the size of the data set *i.e.* if a small data set consists of 10^6 samples then the weight-decay parameter for a data set with 10^7 samples should be 10 times smaller than on the small data set. Similarly, when the minibatch size B is increased the weight-decay parameter λ should be increased by the same factor. The prior norm of the weights is given by $\sqrt{\frac{P}{\lambda}}$ where P is the number of parameters in the neural network. This leads to the fact that, if an ANN with P_1 parameters trained using a minibatch size of B_1 on a data set with N_1 samples with an optimised weight-decay parameter of λ_1 and a constant prior norm of the parameters θ is desired, then the weight-decay parameter λ_2 that we should use for an ANN with P_2 parameters trained using a minibatch size of B_2 on a data set with N_2 samples shall be given by:

$$\lambda_2 = \lambda_1 \frac{N_1}{N_2}\frac{B_2}{B_1}\frac{P_2}{P_1} \tag{2.51}$$

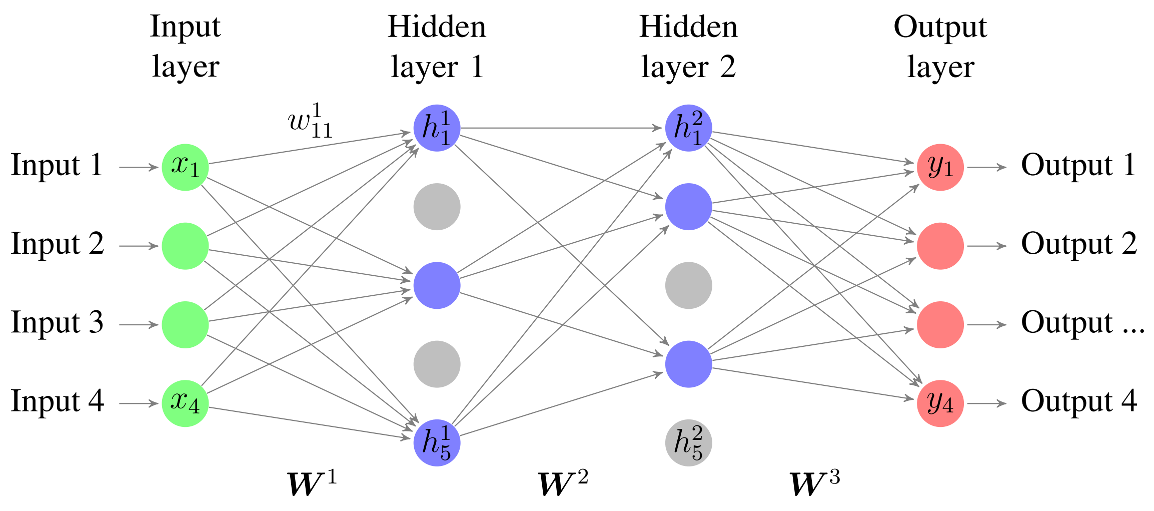


Figure 2.5 Dropout applied to the hidden layers of the neural network of Figure 2.2. Dropped neurons are drawn in grey.

In general, the λ_2 given by this equation will not be the optimal λ to use due to other factors that come into play such as the amount of noise introduced by SGD, which depends on the minibatch size and the learning rate. However, it is a very good starting point for optimising λ.

2.3.2 Dropout

The weight-decay regulariser adds a separate cost function to the original training objective (*e.g.* ML) and there are other regularisers that do the same. Another set of regularisers adds noise to inputs/hidden states or weights of the layers of a ANN. These regularisers fall under the class of stochastic regularisation techniques. These include additive or multiplicative Gaussian noise (Jim et al., 1996) and as the most popular technique *dropout* (Srivastava et al., 2014).

In dropout individual elements of input vectors and hidden vectors are "dropped" meaning that they are set to zero. This is used during training and not during testing. This is equivalent to setting an individual column in a weight matrix to zero. For the input vector $\mathbf{x}^{(l)}$ of each layer l with size $D^{(l)}$ a vector $\boldsymbol{m}^{(l)}$ of the same size is generated. The individual elements of $\boldsymbol{m}^{(l)}$ are each drawn independently from a Bernoulli distribution with $P(m_i^{(l)} = 1) = 1 - p^{(l)}$. $p^{(l)}$ is referred to as the layer-specific *dropout probability* and $\boldsymbol{m}^{(l)}$ as the *dropout mask*, although often the same dropout rate is often applied to all parts of the ANN it has been found that using a larger dropout rate for later layers and a smaller dropout rate for initial layers improves performance (Gal et al., 2017a). Applying the dropout mask then equates to the element-wise product $\tilde{\mathbf{x}}^{(l)} = \mathbf{x}^{(l)} \odot \boldsymbol{m}^{(l)}$.

Dropout attempts to prevent the network from relying on certain features rather than combining multiple features and using the combined statistical strength. At the same time, the ANN effectively has fewer parameters during training, which helps to reduce overfitting as overfitting is generally observed to be more of an issue for models with a larger number of parameters.

2.3.2.1 Dropout at test time

In order to ensure that the magnitude of the hidden vectors is the same with and without dropout, the individual elements of $\tilde{\mathbf{x}}^{(l)}$ are multiplied by $\frac{1}{1-p^{(l)}}$ during training. In other implementations, after training is complete the individual weights of the ANN are divided by $\frac{1}{1-p^{(l)}}$. These are equivalent forms of dropout.

Another option is to train using dropout and then to also test using dropout. To use all the model's parameters multiple dropout masks are sampled and the outputs of the ANN are averaged:

$$\mathbf{y}_i = \frac{1}{J}\sum_{j=1}^{J}\varphi(\mathbf{x}_i, \{\boldsymbol{m}_j\}_{l=0}^{L-1}) \tag{2.52}$$

this process is also called *Monte Carlo Dropout* (Gal and Ghahramani, 2016).

2.4 Uncertainty

In general, it is ideal for any machine learning model not only to give a very good output but also to give a measure of how uncertain the model is about a particular prediction. This is critical in many applications. For example, when an ASR system is used as the input to a Spoken Dialogue System (SDS) it is important for the SDS to be able to realise that it has not fully understood the user and that the user should repeat what she said (Young et al., 2013). If the ASR system only generates its best transcription, this would not be possible.

Predictive uncertainty, which conveys the model's uncertainty in its own output, is usually decomposed into two types of uncertainty. *Epistemic uncertainty*, also called *systematic uncertainty*, arises due to what could be known in theory but is not known in practice (Gal, 2016). In machine learning (especially Bayesian modelling), it represents the uncertainty about the different models that are most suitable to explain the training data and is thus also called *model uncertainty*. In deep learning settings with a rather small data set in comparison to the number of parameters, there will be a large number of different parameter settings that will perfectly explain the training data. In this case, there should be some uncertainty about these parameter settings. Classed under model uncertainty would also be the structure of the machine learning model itself although this is generally not modelled. *Aleatoric uncertainty* also called *statistical uncertainty* or *data uncertainty* captures the noise inherent to the data (Gal, 2016). This can arise due to measurement noise, label noise or class overlap. As a model is trained on more and more data, the epistemic uncertainty reduces. The aleatoric uncertainty, however, is irreducible for the same experimental setup. It can only be reduced by changing the way in which the data was collected by, for example, using better sensors and a more careful labelling procedure.

2.4.1 Predictive Uncertainty in the Bayesian Framework

In the Bayesian framework, the chosen machine learning model will describe a probability distribution $P_\theta(y|\mathbf{x}_i)$ over different classes. A training procedure will infer the distribution $p(\theta|\mathcal{D}_l)$, the posterior over model parameters, the uncertainty of which describes the epistemic uncertainty. In many cases deriving the posterior is not possible and thus *variational inference* is used (Jordan et al., 1999). This means that a distribution $q(\theta)$ is defined, which has a particular form, such as a Gaussian Distribution. During training the parameters of the *variational distribution*, $q(\theta)$, are trained such that it is as similar as possible to $p(\theta|\mathcal{D}_l)$. The predictive distribution $P(y|\mathbf{x}, \mathcal{D}_l)$ can be derived by marginalising over possible parameter settings θ:

$$P(y|\mathbf{x}, \mathcal{D}_l) = \int p(\theta|\mathcal{D}_l) P(y|\mathbf{x}, \theta) d\theta \tag{2.53}$$

$$\approx \int q(\theta) P(y|\mathbf{x}, \theta) d\theta \tag{2.54}$$

Based on the predictive distribution various uncertainty measures can be derived, the most popular of which is the predictive entropy (Shannon, 1948):

$$\mathrm{H}\left[y|\mathbf{x}, \mathcal{D}_l\right] = -\sum_{k=1}^{K} P(y = k|\mathbf{x}, \mathcal{D}_l) \log P(y = k|\mathbf{x}, \mathcal{D}_l) \tag{2.55}$$

2.4.2 Modelling of Uncertainty in Deep Learning

Obtaining a good uncertainty measure is mainly about obtaining a good predictive distribution $P(y|\mathbf{x}, \mathcal{D}_l)$. In the Bayesian framework, this relies on obtaining a good variational distribution $q(\theta)$. Previous sections described how to obtain the (regularised) maximum likelihood estimate θ^{ML} rather than a distribution over possible θ. The maximum likelihood estimate, however, is actually equivalent to $q(\theta) = \delta\left(\theta - \theta^{ML}\right)$, where δ is the continuous Dirac delta function. This yields $P(y|\mathbf{x}, \mathcal{D}_l) = P(y|\mathbf{x}, \theta^{ML})$. Given that $q(\theta)$ contains zero uncertainty the maximum likelihood trained model can only capture aleatoric uncertainty, but not the epistemic uncertainty.

Alternatively, standard maximum likelihood trained models can be explicitly trained to give better uncertainty estimates, though in that case, it is not possible to determine the source of uncertainty.

2.4.3 Predictive Uncertainty using Dropout

In many cases, the integral in Equation (2.54) is intractable, in which case it can be approximated using *Monte Carlo* sampling.

$$\int p(\theta|\mathcal{D}_l)P_\theta(y|\mathbf{x})d\theta \approx \tfrac{1}{J}\sum_{n=1}^{J} P_\theta(y|\mathbf{x},\theta_n), \theta_n \sim q(\theta) \tag{2.56}$$

where J is the number of samples used. The larger S the more lower the variance of this estimate.

If a neural network is trained with Dropout (see Section 2.3.2), then applying a dropout mask can be seen as sampling weight matrices $\boldsymbol{W}^l$ via:

$$\boldsymbol{W}^l = \boldsymbol{M}^l \cdot \text{diag}\left[\{\epsilon_{lj}\}_{j=0}^{D^l-1}\right] \tag{2.57}$$

$$\epsilon_{lj} \sim \text{Bernoulli}\left(p^{(l)}\right), \quad l = 1,\dots,L, j = 0,\dots,D^l - 1 \tag{2.58}$$

here $\boldsymbol{M}^l$ is a trained weight matrix, $\boldsymbol{W}^l$ is the same weight matrix after applying the dropout mask, and D^l is the input dimension of the weight matrix in a neural network with L weight matrices. $q(\theta)$ is the distribution of $\{\boldsymbol{W}^l\}_{l=1}^{L}$. Applying different dropout masks at test time and averaging the resulting output distribuftions results in an approximation to the predictive distribution. It was shown by Gal and Ghahramani (2016) that viewing the dropout mask as sampling from $q(\theta)$ is a theoretically sound interpretation that can be seen as approximate Bayesian variational inference. This view of dropout is useful in a variety of ways which includes the derivation of a method to optimise the layer-specific dropout rates $p^{(l)}$ from Section 2.3.2 by gradient descent (Gal et al., 2017a).

2.5 Summary

This chapter gave a summary of the deep learning theory that will be used throughout this thesis. MLPs will be used in acoustic models of the ASR systems that are used in Chapters 6, 7 and 8. The speaker embeddings model in Chapter 6 uses a TDNN and the active learning method NBest-BALD proposed in Chapter 7 is further verified with a TDNN-based acoustic model. The Emformer, the memory efficient alternative to the transformers, is used for the acoustic model in Chapter 9. The introduction to uncertainty in Section 2.4 is an important inspiration for the work in Chapter 7. Further, Sections 3.10 and 3.11 will explain how the building blocks presented in this chapter can be used to directly model sequences.

Chapter 3

Speech Recognition

Automatic Speech Recognition (ASR) is defined as the automatic process of converting spoken language into the corresponding word-level transcription by a computer.

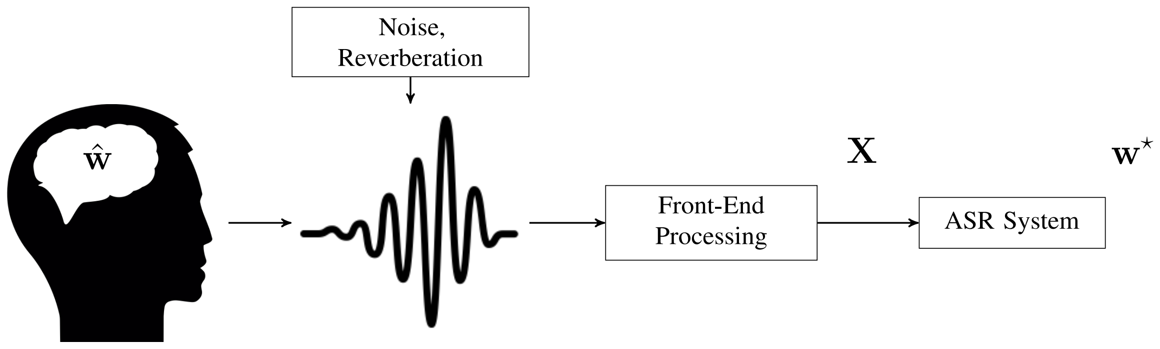


Figure 3.1 Speech production and recognition. $\hat{\mathbf{w}}$ is the correct hypothesis. $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_T]$ is the input feature vectors to the ASR system and $\mathbf{w}^\star = [w_1^\star, ..., w_M^\star]$ is the output of the ASR system.

Figure 3.1 illustrates the problem statement of ASR. A human says the utterance $\hat{\mathbf{w}}$ and produces a waveform that is then corrupted by noise and/or reverberations. A front-end processing module, which includes not only a microphone but also digital signal processing (DSP) algorithms, produces the input sequence of length T,

$$\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_T] \tag{3.1}$$

that is passed to the ASR system which outputs the hypothesis of length M,

$$\mathbf{w}^\star = [w_1^\star, ..., w_M^\star], w_i \in [1, 2, ..., \|V\|] \tag{3.2}$$

Ideally, $\hat{\mathbf{w}} = \mathbf{w}^\star$. ASR is a difficult problem because different people generate different waveforms for the same $\hat{\mathbf{w}}$ and even those for the same speaker will be different every time. Furthermore, the recording environment also affects $\mathbf{X}$, such as the aforementioned noise and reverberation, but also the codec being used or transmission errors, such as packet loss. Another challenge is that the input sequence is much longer than the output sequence. All models that try to solve the problem of ASR have some mechanism built-in that attempts to align parts of the input and parts of the output.

Given the inherent noise in the process $\hat{\mathbf{w}} \rightarrow \mathbf{X}$ it is advantageous to use a probabilistic framework *i.e.* for the ASR system to build a model $P_\theta(\mathbf{w}|\mathbf{X})$. The ASR system then generates an output using:

$$\mathbf{w}^\star = \arg\max_{\mathbf{w}} P_\theta(\mathbf{w}|\mathbf{X}) \tag{3.3}$$

where $\mathbf{w}^\star$ is the most probable *hypothesis* for the transcription given the speech waveform. Broadly speaking, there are two approaches in the probabilistic framework. The first approach is to directly model $P(\mathbf{w}|\mathbf{X})$ in a discriminative fashion. Often this is done using a single end-to-end differentiable Artificial Neural Network (ANN). Examples for this are Attention-Based Encoder-Decoder (AED) models (Chan et al., 2016; Chiu et al., 2018; Chorowski et al., 2015) and transducer models (Graves, 2012). The second approach is to apply Bayes' rule and to use two separate models, in a generative fashion

$$P_\theta(\mathbf{w}|\mathbf{X}) = \frac{p_\theta(\mathbf{X}|\mathbf{w})P_\theta(\mathbf{w})}{p_\theta(\mathbf{X})} \propto p_\theta(\mathbf{X}|\mathbf{w})P_\theta(\mathbf{w}) \tag{3.4}$$

where $p_\theta(\mathbf{X}|\mathbf{w})$ is referred to as the Acoustic Model (AM) and $P_\theta(\mathbf{w})$ is referred to as the Language Model (LM). Hidden Markov Model (HMM)-based models (Gales and Young, 2008; Rabiner, 1989) have been the most prominent approach to the generative approach.

This chapter explains how to model the ASR system first and then how to train it. Section 3.1 describes how the sequence of vectors $\mathbf{X}$ can be obtained from the audio. Section 3.2 describes how the word sequence $\mathbf{w}$ can be represented in terms of smaller sub-word units. Sections 3.3 and 3.4 describe how the AM and LM are modelled, respectively. Section 3.5 then discusses how to obtain $\mathbf{w}^\star$ from $p_\theta(\mathbf{X}|\mathbf{w})P_\theta(\mathbf{w})$, which is called decoding. Section 3.6 then explains how the ASR system is evaluated based on the outputs discussed in Section 3.5. Section 3.7 expands upon Section 3.5 and describes different ways in which multiple competing hypotheses can be represented and used, instead of just the single best hypothesis $\mathbf{w}^\star$. Section 3.8 describes how to measure the confidence the system has in its output. Section 3.9 then explains how to train the AM. The previous sections mainly focused on models with a separate AM and LM whilst Sections 3.10 and 3.11 describe methods that can model $P_\theta(\mathbf{w}|\mathbf{X})$ directly in one single model.

3.1 Input Representations

This section is concerned with obtaining the series of vectors

$$\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_T] \tag{3.1}$$

from the waveform representation. This process is also referred to as *feature extraction*. Whilst it is possible to use the waveform directly to as the input representation (Sainath et al., 2015; Tüske et al., 2014; von Platen et al., 2019), this section focuses on turning a waveform into a series of vectors. This section also ignores any noise-cancelling algorithms, packet loss recovery algorithms or the merging of multiple microphones.

The acoustic features are expected to carry sufficient information for ASR, *i.e.* not discard any information that is useful for the transcription $\hat{\mathbf{w}}$ of the waveform. At the same time, it is preferable for the acoustic features to be robust to any changes that are not related to the transcription such as the microphone, the channel, the background noise, the intensity with which the speaker speaks or the codec with which the waveform was encoded. When choosing a certain feature extraction method it is important to keep in mind the type of ASR model being used later. Some approaches, such as the Gaussian Mixture Model (GMM)-HMM have much stronger assumptions about the input vectors $\mathbf{x}_i$ than theANN-HMM. It is important to match the characteristics of the extracted features to the assumptions of the ASR model that is being used.

In most feature extraction methods, the waveform is first split into *speech frames*. Each frame is usually 25 ms long. One frame is extracted strictly every 10 ms, such that there is an overlap between the frames of 15 ms. The total number of frames is equal to T, the length of $\mathbf{X}$.

One type of feature being used in this thesis are Filter Bank (FBANK) features. The Discrete Fourier Transform (DFT) is applied to each frame to obtain a frequency representation of the frame. To reduce distortion, the frame is first windowed (usually using a Hamming window). Only the magnitude coefficients from the DFT are then considered. Usually, the energies are used *i.e.* the magnitude coefficients squared. These are weighted and summed using a desired number (D) of triangular filters. D will be the size of the final feature vectors $\mathbf{x}_t$. The triangular filters are overlapping and are spaced non-linearly based on the sensitivity of the human ear[1]. The centres of the triangular filters are spaced linearly on the mel-scale. This means that, along the linear Hz frequency scale, they are spaced roughly linearly for lower frequencies and logarithmically for higher frequencies. The mel-scale to Hz-scale conversion

[1]At lower frequencies, it is easier to discern differences in frequencies than at higher frequencies.

(and vice-versa) is given by:

$$\begin{aligned} f_{mel} &= 2595 \log_{10}\left(1 + \frac{f_{Hz}}{700}\right) \\ f_{Hz} &= 700\left(10^{f_{mel}/2595} - 1\right) \end{aligned} \tag{3.5}$$

The final step to obtain the FBANK features is to apply the log function to the values produced by each filter. Many other processes can be included when calculating FBANK features, such as applying a high/low frequency cut-off after the DFT.

To obtain Mel-scale Frequency Cepstral Coefficients (MFCCs) (Davis and Mermelstein, 1980) features, a truncated Discrete Cosine Transform (DCT) is applied to the FBANK features.

$$x_i^M = \sqrt{\frac{2}{D_F}} \sum_{j=0}^{D_F-1} x_j^F \cos\left[\frac{i \cdot \left(j - \frac{1}{2}\right)}{D_F}\right], i = 1, \ldots, D_M - 1. \tag{3.6}$$

where $\mathbf{x}^F$ are the FBANK coefficients of dimensionality D_F and the MFCC features (cepstral features) $\mathbf{x}^M$ are of size D_M. x_0^M is often set to a measure of the signal energy in the frame. The DCT (if truncated *i.e.* $D_M < D_F - 1$) can strongly decorrelate the coefficients in $\mathbf{x}$ and hence allows them to be modelled using Gaussian distributions with diagonal covariance matrices. Sinusoidal liftering (filtering in the cepstral domain) can be applied to the MFCCs to increase the dynamic range of higher order MFCCs.

Another feature type used in this thesis are Perceptual Linear Prediction (PLP) features Hermansky (1990). PLP features are, like MFCCs, cepstral features. First, the DFT is computed to obtain the power spectrum (square of the magnitudes). The power spectrum is then warped using a Bark-Hertz transformation (Zwicker, 1961) (similar to mel-scale) and then pre-emphasized based on an equal-loudness curve to take into account that hearing sensitivity is different at different frequencies (Robinson and Dadson, 1956). Then the amplitude of each frequency-power coefficient is compressed using the cube-root (similar to the log compression used in MFCCs). This takes into account the non-linear relationship between the intensity of sound and perceived loudness (Stevens, 1957). Then the inverse DFT is applied to obtain the cepstrum, which is then approximated using a linear predictive analysis and cepstral transform applied. PLP features, similar to MFCC features, are de-correlated and can be modelled with diagonal covariance matrices.

3.1.1 Deltas and Normalisation

MFCC and PLP features are often expanded using their time derivatives (Furui, 1986). Not only the first derivative but also the second and third derivatives can be used. These derivatives and delta coefficients are normally calculated based on the least-square linear-regression formula:

$$\mathbf{x}'_t = \frac{\sum_{i=1}^{\Theta} i \cdot \left(\mathbf{x}_{t+i} - \mathbf{x}_{t-i} \right)}{2 \sum_{i=1}^{\Theta} i^2} \tag{3.7}$$

The same formula is used when calculating the second and third order derivatives $\mathbf{x}''$ and $\mathbf{x}'''$, by replacing $\mathbf{x}$ with $\mathbf{x}'$ and $\mathbf{x}''$, respectively. $\Theta = 2$ is a popular parameter choice for the "delta window".

For both GMM-HMM and ANN-HMM models, it is useful to normalise the input sequence $\mathbf{X}$. This is done by first subtracting the mean-vector and then dividing by the standard deviation. This is done separately for each dimension. The mean-vector used for normalisation is often estimated on an utterance basis such that there is one mean-vector per utterance. The variance-vector is often estimated using more data *e.g.* globally, such that there is one variance-vector for the entire data set or on the conversation level, such that there is one variance-vector for each conversation within the data set.

3.2 Output Units

This section is concerned with how to represent the output sequence,

$$\mathbf{w} = [w_1, ..., w_M, w_m] \in [1, 2, ..., \|V\|] \tag{3.2}$$

It is often impractical to directly model the word sequence $\mathbf{w}$, because the size of the vocabulary $\|V\|$ might be too large. Also, some words might be contained in the vocabulary V that are not contained in the training data. Hence, each spoken word w_i is usually composed into a sequence of *subword units* $\mathbf{q}^{w_m} = [q_1, ..., q_{M_{w_i}}]$. If there are multiple possibilities for $\mathbf{q}^{w_i}$ the AM probability becomes,

$$p(\mathbf{X}|\mathbf{w}) = \sum_{q} P(\mathbf{q}|\mathbf{w}) p(\mathbf{X}|\mathbf{q}) \tag{3.8}$$

where the distribution $P(\mathbf{q}|\mathbf{w})$ is

$$P(\mathbf{q}|\mathbf{w}) = \prod_{m=1}^{M} P(\mathbf{q}^{w_m}|w_m) \tag{3.9}$$

where the distribution $P(\mathbf{q}^{w_m}|w_m)$ is also referred to as the *pronunciation dictionary*. It contains all possible mappings of $\mathbf{w}$ into subword units. English pronunciation dictionaries typically contain fewer than two non-zero probabilities per w_m on average. Some possible subword units are depicted in Table 3.1. After training the subword AM $p(\mathbf{X}|\mathbf{q})$, it can be used together with $P(\mathbf{q}^{w_m}|w_m)$ to recognise words that are in the pronunciation dictionary but may not be in the training data.

words	(silence)		stop		that		(silence)	
mono-grapheme	s	t	o	p	t	h	a	t
tri-grapheme	sil-s+t	s-t+o	t-o+p	o-p+t	p-t+h	t-h+a	h-a+t	t-h+sil
monophone	s	t	aa	p		dh	ae	t
logical triphone	sil-s+t	s-t+aa	t-aa+p	aa-p+dh		p-dh+ae	dh-ae+t	ae-t+sil
physical triphone	m1398	m308	m633	m1316		m1485	m1991	m1811
word pieces	sto		p</w>		that</w>			
	sto		p</w>		th		at</w>	

Table 3.1 Different modelling units for the phrase “stop that”. The symbol before ‘-’ is the left context and the symbol after ‘+’ is the right context.

The simplest set of subword units are *graphemes* (in Table 3.1 labelled as mono-graphemes). Graphemes are the smallest functional unit of a writing system. In an alphabetic writing system such as English, these are the letters (as in Table 3.1). In a logographic system, such as Chinese, graphemes would be the characters representing words or morphemes[2] and in a system using a syllabary, such as Japanese kana, they are the syllabic characters representing different syllables. The set of graphemes can be extended based on their positions within words (Gales et al., 2015; Le et al., 2019). An advantage of using graphemes is that the pronunciation dictionary $P(\mathbf{q}^{w_m}|w_m)$ only contains one $\mathbf{q}^{w_m}$ for each w_m and hence no sum is required in Equation (3.8). An issue with graphemes can be that in some languages the graphemic transcription does not correspond well with the phonetic transcription *i.e.* the same grapheme is pronounced differently in different words (*e.g.* “through”, “though”, “thought”, “tough”).

Therefore, using base phones as the set of subword units is popular (monophone in Table 3.1). Since there is a more direct relationship between the phonetic transcription and the spoken word, the distribution $p(\mathbf{X}|\mathbf{q})$ is easier to model. This is because the pronunciation dictionary takes care of modelling the irregularities of the graphemic to acoustic realisation.

Another option for the sub-word units are word pieces. Frequent groups of graphemes are represented as one unit (*e.g.* “the”, “ing”) and ends of words are also represented as a separate unit (the </w> in Table 3.1). Byte Pair Encoding (BPE) (Gage, 1994) is a popular algorithm to

[2]In English the ampersand "&" is also a logogram.

create a set of word pieces. Word pieces are more popular in discriminative modelling than in generative approaches.

Tri-graphemes and triphones will be explained in Section 3.3.3.

3.3 Acoustic Modelling

This section is concerned with the acoustic model $p(\mathbf{X}|\mathrm{w})$. In particular, how to model the acoustic model based on subword units *i.e.* $p(\mathbf{X}|\mathbf{q})$ in Equation (3.8). In this section, the term AM will mostly refer to $p(\mathbf{X}|\mathbf{q})$ rather than $p(\mathbf{X}|\mathrm{w})$. To obtain $p(\mathbf{X}|\mathrm{w})$, Equation (3.8) can be used. The subword units q_m will often be referred to as *phones* even though graphemes could also be used.

3.3.1 Hidden Markov Models

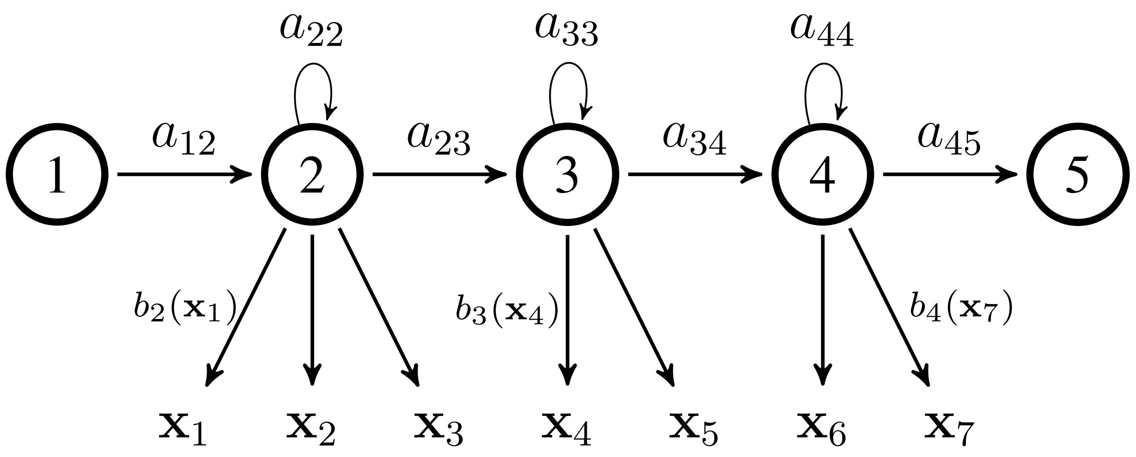


Figure 3.2 A three state HMM modelling the emission of seven acoustic vectors $\mathbf{x}_{1:7}$. a_{ij} is the probability of transitioning to state j when being in state i. $b_i(\mathbf{x}_t)$ is the probability of emitting vector $\mathbf{x}_t$ when in state i. a_{ij} is a discrete probability, while $b_i(\mathbf{x}_t)$ is a continuous probability density.

Within the AM, $p(\mathbf{X}|\mathbf{q})$, each phone q is represented by a Hidden Markov Model (HMM). The distribution $p(\mathbf{X}|\mathbf{q})$ models how the frames $\mathbf{x}_t$ are generated/emitted. Each HMM consists of multiple states; often 5 states of which the first and last state are non-emitting states, meaning that if the process is in those states, no feature vector is emitted. The number of time steps spent in emitting states is equal to the total number of emitted acoustic vectors. Figure 3.2 depicts an HMM with three emitting states that is used to model the emission of seven acoustic vectors. In an HMM the state-sequence $\mathbf{s}$ is modelled as a *Markov chain*. This means that the probability of being in state s_t at time t given all previous states uses the following independence assumption:

$$P(s_t = j|s_{1:t-1}) = P(s_t = j|s_{t-1} = i) = a_{ij}, \quad \sum_{j=1}^{S} a_{ij} = 1 \, \forall \, i = 1, \ldots, S \tag{3.10}$$

where a_{ij} is the transition probability and S is the total number of HMM-states. The emissions have the following independence assumption:

$$p(\mathbf{x}_t|s_{1:t}, \mathbf{x}_{1:t-1}) = p(\mathbf{x}_t|s_t) = b_{s_t}(\mathbf{x}_t) \tag{3.11}$$

where $b_i(\mathbf{x}_t)$ is the emission probability *i.e.* the probability of emitting vector $\mathbf{x}_t$ when being in state $s_t = i$.

To summarise, in an HMM, states are conditionally independent of all other states given the previous state, and acoustic vectors are conditionally independent of all other observations and states given the state that generated the vector. While the state-transition probabilities $a_{s_{t-1}s_t}$ are simple categorical distributions, the emission probability $b_{s_t}(\mathbf{x}_t)$ can take a large range of forms that are described in Section 3.3.4.

To model the sequence of subword units $\mathbf{q}$, all HMMs (with their associated parameters) of the individual phones in $\mathbf{q}$ are strung together by connecting them at their start and end non-emitting states. This combined HMM is called the *composite* HMM. This is illustrated in Figure 3.3 where three constituent HMMs are combined to one composite HMM.

There is a large set of possible state-sequences ($\mathbf{s} \in \mathcal{S}$) that can be taken through the composite HMM, hence $p(\mathbf{X}|\mathbf{q})$ is given by,

$$p(\mathbf{X}|\mathbf{q}) = \sum_{\mathbf{s}\in\mathcal{S}} P(s_{1:T}, \mathbf{x}_{1:T}|\mathbf{q}) \tag{3.12}$$

where $P(s_{1:T}, \mathbf{x}_{1:T}|\mathbf{q})$ is modelled by combining Equation (3.10) with Equation (3.11),

$$P(s_{1:T}, \mathbf{x}_{1:T}|\mathbf{q}) = a_{s_0 s_1} \prod_{t=1}^{T} a_{s_t s_{t+1}} b_{s_t}(\mathbf{x}_t) \tag{3.13}$$

where s_0 and s_{T+1} would be set to the start and end non-emitting states of the composite HMM.

3.3.2 Forward-Backward Algorithm

The acoustic likelihood $p(\mathbf{X}|\mathbf{q})$ requires the summation over all possible state-sequences $\mathbf{s} \in \mathcal{S}$ according to Equation (3.12). The number of possible state-sequences scales with $\mathcal{O}(S^T)$.

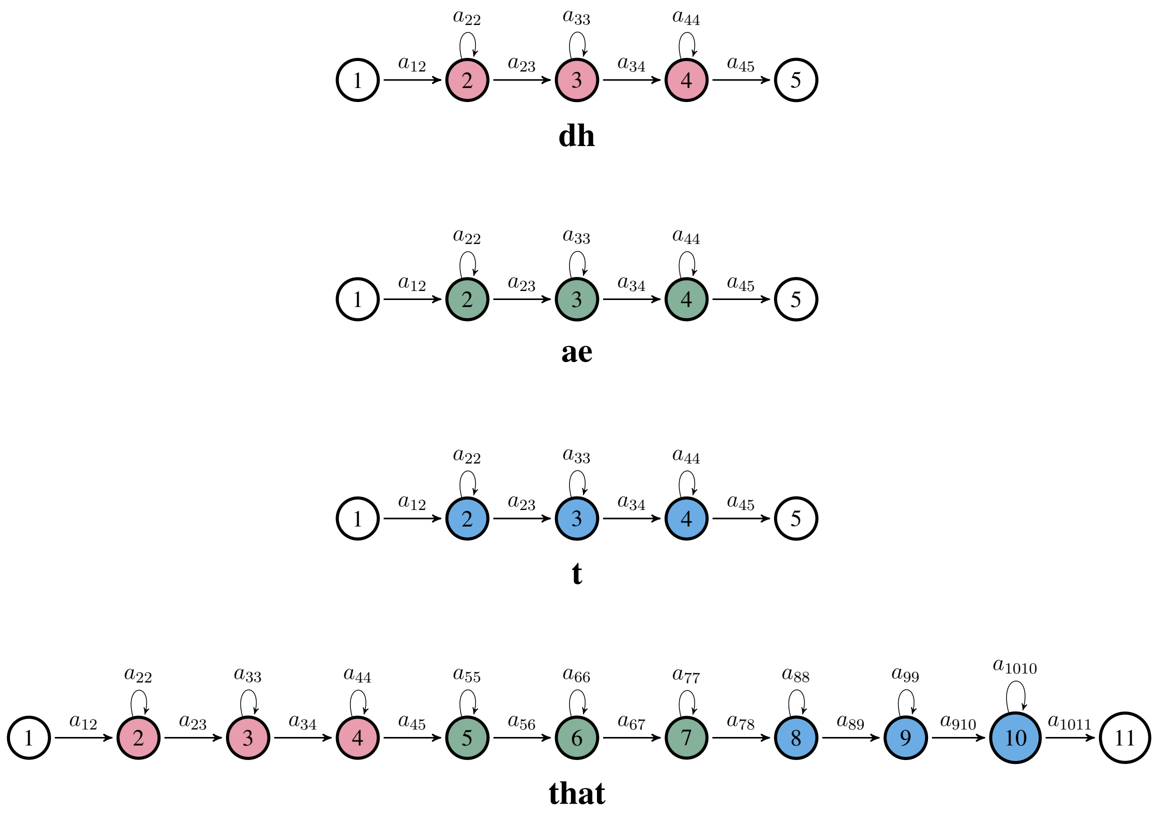


Figure 3.3 The composite HMM for the transcription ‘that’. Composing $\mathbf{q}$ based on constituent HMMs. The three 3-state HMMs for the phones ‘dh’, ‘ae’ and ‘t’ that constitute the pronunciation of ‘that’ are combined to the composite HMM that represents ‘that’.

Using the forward-backward algorithm, the time complexity of the likelihood calculation can be reduced to $\mathcal{O}(ST)$.

The forward-backward algorithm is based on the recursive calculation of two variables. The forward variable $\alpha_i(t)$,

$$\alpha_i(t) = p\left(\mathbf{x}_{1:t}, s_t = i|\mathbf{q}\right) \tag{3.14}$$

and the backward variable $\beta_i(t)$,

$$\beta_i(t) = p\left(\mathbf{x}_{t+1:T}|s_t = i, \mathbf{q}\right) \tag{3.15}$$

Multiplying the two variables yields,

$$\alpha_i(t) \cdot \beta_i(t) = p(\mathbf{X}, s_t = i|\mathbf{q}) \tag{3.16}$$

Marginalising over all possibilities for s_t yields the likelihood,

$$p(\mathbf{X}|\mathbf{q}) = \sum_i p(\mathbf{X}, s_t = i|\mathbf{q}) = \sum_i \alpha_i(t) \cdot \beta_i(t) \tag{3.17}$$

which is independent of t and can be calculated purely based on $\alpha_i(T)$or $\beta_i(0)$. The forward variable $\alpha_i(t)$ is calculated recursively using $\alpha_j(t-1)$ using the equation:

$$\begin{aligned}
\alpha_i(t) &= p\left(\mathbf{x}_{1:t}, s_t = i|\mathbf{q}\right) \\
&= p\left(\mathbf{x}_t, \mathbf{x}_{1:(t-1)}, s_t = i|\mathbf{q}\right) \\
&= p\left(\mathbf{x}_{1:(t-1)}, s_t = i|\mathbf{q}\right) \cdot p\left(\mathbf{x}_t|\cancel{\mathbf{x}_{1:(t-1)}}, s_t = i, \mathbf{q}\right) \\
&= \left(\sum_j p\left(\mathbf{x}_{1:(t-1)}, s_t = i, s_{(t-1)} = j|\mathbf{q}\right)\right) \cdot b_i(\mathbf{x}_t) \\
&= \left(\sum_j p\left(\mathbf{x}_{1:(t-1)}, s_{(t-1)} = j|\mathbf{q}\right) P\left(s_t = i|\cancel{\mathbf{x}_{1:(t-1)}}, s_{(t-1)} = j, \mathbf{q}\right)\right) \cdot b_i(\mathbf{x}_t) \\
&= \left(\sum_j \alpha_j(t-1) a_{ji}\right) \cdot b_i(\mathbf{x}_t)
\end{aligned} \tag{3.18}$$

where the strikeouts $\cancel{\mathbf{x}_{1:(t-1)}}$ are based on the independence assumptions of the HMM.

The backward variable $\beta_i(t)$ is calculated recursively using $\beta_j(t+1)$ using the equation:

$$\begin{aligned}
\beta_i(t) &= p\left(\mathbf{x}_{t+1:T}|s_t = i, \mathbf{q}\right) \\
&= p\left(\mathbf{x}_{t+1}, \mathbf{x}_{t+2:T}|s_t = i, \mathbf{q}\right) \\
&= \sum_j \left(p\left(\mathbf{x}_{t+1}, \mathbf{x}_{t+2:T}, s_{t+1} = j|s_t = i, \mathbf{q}\right)\right) \\
&= \sum_j \Bigg(P\left(s_{t+1} = j|s_t = i, \mathbf{q}\right) \cdot p\left(\mathbf{x}_{t+1}|\cancel{s_t = i}, s_{t+1} = j, \mathbf{q}\right) \\
&\qquad \cdot p\left(\mathbf{x}_{t+2:T}|\cancel{\mathbf{x}_{t+1}, s_t = i}, s_{t+1} = j, \mathbf{q}\right)\Bigg) \\
&= \sum_j \left(a_{ij} \cdot b_j(\mathbf{x}_{t+1})\beta_j(t+1)\right)
\end{aligned} \tag{3.19}$$

The two calculations are initialised using:

$$\alpha_i(0) = 1 \text{ for } i = 1 \text{ and } 0 \text{ otherwise} \tag{3.20}$$

$$\beta_i(T) = a_{iS} \tag{3.21}$$

For the recursive calculations, numerical underflow can occur, and therefore log-arithmetic is normally used.

3.3.2.1 State occupancy and state-transition probabilities

The forward and backward variables can be used to calculate the likelihood Equation (3.17). They can also calculate two other probabilities that are important in the rest of this chapter. The first is the *state occupancy* $\gamma_i(t)$, which is the probability that the acoustic vector $\mathbf{x}_t$ was emitted by state i:

$$\begin{aligned}\gamma_i(t) &= P(s_t = i|\mathbf{X}, \mathbf{q}) \\ &= \frac{P(s_t = i, \mathbf{X}|\mathbf{q})}{p(\mathbf{X}|\mathbf{q})} \\ &= \frac{\alpha_i(t)\beta_i(t)}{\sum_j \alpha_j(t)\beta_j(t)}\end{aligned} \tag{3.22}$$

The second is the *state-transition posterior* $\xi_{ij}(t)$, which is the probability that the acoustic vectors $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$ where emitted from states i and j, respectively.

$$\begin{aligned}\xi_{ij}(t) &= P(s_t = i, s_{t+1} = j|\mathbf{X}, \mathbf{q}) \\ &= \frac{P(s_t = i, s_{t+1} = j, \mathbf{X}|\mathbf{q})}{p(\mathbf{X}|\mathbf{q})} \\ &= \frac{a_{ij}\alpha_i(t)\beta_i(t+1)b_j(\mathbf{x}_{t+1})}{\sum_k \alpha_k(t)\beta_k(t)}\end{aligned} \tag{3.23}$$

3.3.3 Co-articulation and parameter tying

Individual sub-word units q_i will be pronounced differently depending on the surrounding sub-word units $\{q_{i-1}, q_{i+1}\}$. This is especially true for graphemes, but also when q_i is a phone. When using graphemes, this is simply part of the irregularities of the graphemic to acoustic realisation (*e.g.* ‘t’ and ‘h’ in “to hold” or in “that”). When q_i is a phone, this effect is called *co-articulation*. It basically means that properties of one phone spread to its neighbouring phones. During the production of speech, any particular articulator (*e.g.* tongue, lips, teeth, and hard palate) is required to go through a rapid sequence of movements. Therefore the place to which an articulator moves in order to produce a certain sound is influenced by its previous position and its following position (Ashby and Maidment, 2005). For example, the /p/ in “stop that” has its burst suppressed by the following consonant.

The above insight from speech science mandates that q_i should be modelled differently depending on the surrounding sub-word units $\{q_{i-1}, q_{i+1}\}$. This is done by changing the choice of sub-word unit from monophones to triphones (Schwartz et al., 1985) (recall Table 3.1, row "logical triphone"). The triphone is a model for a phone in the context of its preceding and succeeding phones ($\{q_{i-1}, q_{i+1}\}$). Hence, for the word "stop", the second sub-word unit is 's-t+aa' instead of 't'. This is also referred to as using a *context-dependent* HMM.

Other options are left- and right-biphones (using either $\{q_{i-1}$ or $q_{i+1}\}$ as context) and even up to pentaphone HMMs (using $\{q_{i-2:i-1}$ or $q_{i+1:i+2}\}$ as context). The simple systems introduced before are referred to as *context-independent* HMMs or *monophone* HMMs. Instead of viewing this from the perspective of speech science there is also a mathematical interpretation, which is that context-dependent HMMs relax the Markov assumptions of Equations (3.10) and (3.11) as it conditiones the probabilities $P(s_t|s_{t-1}, q_i)$ and $p(\mathbf{x}_t|s_t, q_i)$ on the surrounding phones. Relaxing these assumptions directly leads to larger modelling capabilities.

For a triphone model that is modelling a set of Q sub-word units, this results in Q^3 different HMMs. For most standard systems with often $Q \approx 50$ this becomes too many HMM-parameters for practical amounts of data. To avoid the resulting data sparsity problems, the complete set of Q^3 triphones, called *logical* triphones is mapped to a reduced set of *physical* triphones (see Table 3.1 row "physical triphone"). The set of logical triphones that map to the same physical triphone will then share the parameters associated with $P(s_t|s_{t-1}, q_i)$ and $p(\mathbf{x}_t|s_t, q_i)$. For example, the physical triphone 'm633' in Table 3.1 could be shared between the 't-aa+p' in "stop" and the 'ch-aa+p' in "chop". In practice, even more parameter tying is used, which is tying at the state-level. Often state 2 of triphones with the same left-context or state 4 of triphones with the same right-context can be tied. This means that these tied states will share the same parameters associated with the emission probability $p(\mathbf{x}_t|s_t, q_i)$. For example, state 2 of the HMM of 't-aa+p' in 'stop' can be shared with the HMM of 't-aa+m' in "Tom" and state 4 can be shared with the HMM of 'k-aa+p' in "copper".

The choice of which states to tie is commonly made using decision trees (Young et al., 1994). The decision tree approach is actually a top-down approach, rather than the bottom-up approach just described, in which the set of context-dependent states corresponding to the same context-independent state is split at each node of the decision tree. The context-dependent states that end up in the same leaf node are then tied together. The total set of questions is often rule-based, using linguistic knowledge, whilst whether or not to apply each of them is decided in a data-driven manner. The decision tree approach has the advantage that states the physical state corresponding to a logical state unseen during training can be simply computed. This is important for systems with a very large set of logical states for which these mappings cannot be stored. The set of physical triphones is in the end determined by merging those logical triphones that have all their states shared between each other.

3.3.4 Modelling of Emission Probabilities

The section is about how to model the emission probabilities $b_i(t) = p(\mathbf{x}_t|s_t = i)$ (Equation (3.11)).

3.3.4.1 Gaussian Mixture Models

The traditional approach to model the emission probabilities uses GMMs (Juang, 1985):

$$p(\mathbf{x}_t|s_t = i) = \sum_{k=1}^{K} \pi_{ik} \cdot \mathcal{N}(\mathbf{x}_t; \boldsymbol{\mu}_{ik}, \Sigma_{ik}) \tag{3.24}$$

where the mixture component priors π_{ik} (the weight of the Gaussian k in state i) satisfy $\sum_k \pi_{ik} = 1$ and $\mathcal{N}(\cdot)$ is the Normal distribution with mean $\boldsymbol{\mu}_{ik}$ and co-variance matrix Σ_{ik} such that,

$$\mathcal{N}(\mathbf{x}_t; \boldsymbol{\mu}_{ik}, \Sigma_{ik}) = \frac{1}{\sqrt{(2\pi)^D |\Sigma_{ik}|}} \exp\left(-\frac{1}{2}(\mathbf{x}_t - \boldsymbol{\mu}_{ik})^T \Sigma_{ik}^{-1} (\mathbf{x}_t - \boldsymbol{\mu}_{ik})\right) \tag{3.25}$$

Given enough components and no restrictions placed onto the covariance matrix GMMs can model any probability distribution to any required level of accuracy. When $\mathbf{x}_t$ is a high-dimensional vector using a full covariance matrix for each Gaussian component, for every physical state, leads to data sparsity issues. In practice only diagonal or sometimes block diagonal matrices are used. In these cases, it is necessary to use input features that are decorrelated such as the PLP or MFCC features of Section 3.1.

3.3.4.2 ANN-HMM hybrid models

Another option for modelling emission probabilities of HMMs uses ANNs (see Chapter 2). In this approach, an ANN is used to classify the individual observation vectors $\mathbf{x}_t$ into the different physical states and then Bayes' rule is applied:

$$p(\mathbf{x}_t|s_t = i) = \frac{P(s_t = i|\mathbf{x}_t)p(\mathbf{x}_t)}{P(s_t = i)} \tag{3.26}$$

where $P(s_t = i|\mathbf{x}_t)$ is given by the i-th output of a neural network $\varphi(\mathbf{x}_t)$,

$$P(s_t = i|\mathbf{x}_t) = [\mathbf{y}_t]_i = [\varphi(\mathbf{x}_t)]_i \tag{3.27}$$

This is illustrated in Figure 3.4 where the connections between a HMM and a Deep Neural Network (DNN) are shown. The width of the output layer is the number of physical states (see Section 3.3.3).

Since $p(\mathbf{x}_t)$ does not influence the transcription, it can therefore be ignored. The $P(s_t = i)$ probabilities are unigram probabilities that can be estimated from data. In practice, it is assumed that this probability is constant w.r.t. t and hence the approximation $P(s_t = i) \mathrel{\hat{=}} P(s = i)$ is used. To determine $P(s = i)$, $P(s_t = i|\mathbf{x}_t)$ can be averaged through the training set (Manohar et al., 2015):

$$\begin{aligned} P(s = i) &\approx \frac{1}{N}\sum_{n=1}^{N}\frac{1}{T_n}\sum_{t=1}^{T_n} P(s_{n,t} = i|\mathbf{x}_n) \\ &= \frac{1}{N}\sum_{n=1}^{N}\frac{1}{T_n}\sum_{t=1}^{T_n} \left[\varphi\left(\mathbf{x}_{n,t}\right)\right]_i \end{aligned} \tag{3.28}$$

where $s_{n,t}$ is the state in utterance n at time step t and $\mathbf{x}_{n,t}$ is an acoustic vector of utterance n at time step t.

Otherwise, the sequence of HMMs states that represents the most probable path through the concatenated sub-word unit HMMs can be used. The alignment is calculated using the Viterbi algorithm (Viterbi, 1967) (Section 3.5.1). Then state-level unigram probabilities can be estimated using a standard frequency count. In practice, and in this thesis, this method was used. The first approach of averaging the state-level posterior probabilities is used less often (Manohar et al., 2015).

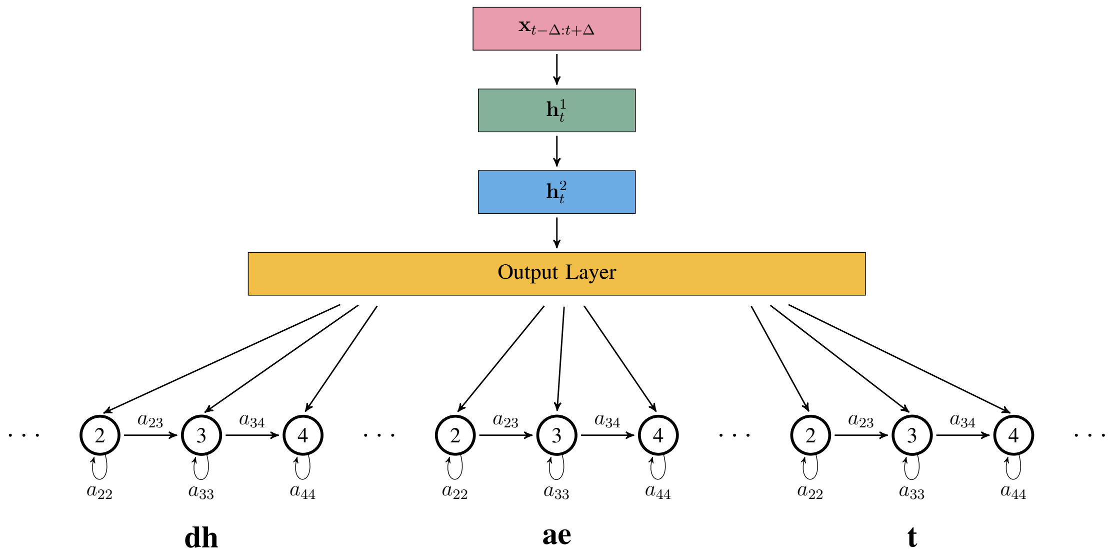


Figure 3.4 A diagram of an ANN-HMM hybrid model showing the relationship between the output probabilities of the ANN and the emission probabilities of the HMM-states.

In Equation (3.27), the input to the ANN that produces the state posteriors is $\mathbf{x}_t$. If an Multilayer Perceptron (MLP) is used, the input to the MLP is often expanded to include a certain number of previous and future context frames *i.e.*

$$P(s_t = i|\mathbf{x}_t) = [\mathbf{y}_t]_i = \left[\varphi^{\text{MLP}}\left(\mathbf{x}_{t-\Delta_1:t+\Delta_1}\right)\right]_i \tag{3.29}$$

where $\Delta_1 = 3$ was often used in this thesis. When Recurrent Neural Networks (RNNs) are used effectively, all previous frames are used and sometimes the output after some future frame is used,

$$P(s_t = i|\mathbf{x}_t) = [\mathbf{y}_t]_i = \left[\varphi^{\text{RNN}}\left(\mathbf{x}_{1:t+\Delta_2}\right)\right]_i \tag{3.30}$$

where $\Delta_2 = 5$ is a typical value. For reasons, such as better shuffling and reduced memory requirements, an RNN that is unfolded (see Figure 2.4b) for a fixed number of time-steps is often used instead of using the full history (Saon et al., 2014),

$$P(s_t = i|\mathbf{x}_t) = [\mathbf{y}_t]_i = \left[\varphi^{\text{RNN}_{unf}}\left(\mathbf{x}_{t-\Delta_3:t+\Delta_4}\right)\right]_i \tag{3.31}$$

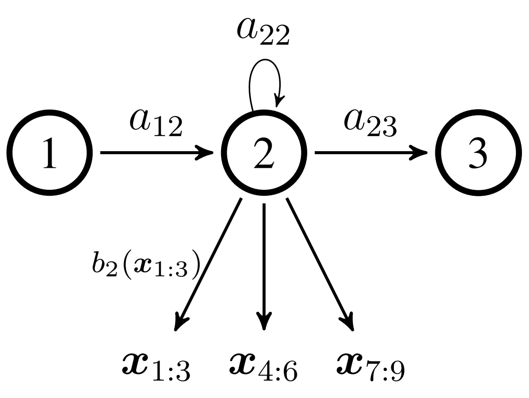
Usually $\Delta_3 \gg \Delta_4$ such as $\Delta_3 = 15$ and $\Delta_4 = 5$. In the Time Delay Neural Network (TDNN) of Figure 2.3,

$$P(s_t = i|\mathbf{x}_t) = [\mathbf{y}_t]_i = \left[\varphi^{\text{TDNN}}\left(\mathbf{x}_{t-13:t+9}\right)\right]_i \tag{3.32}$$

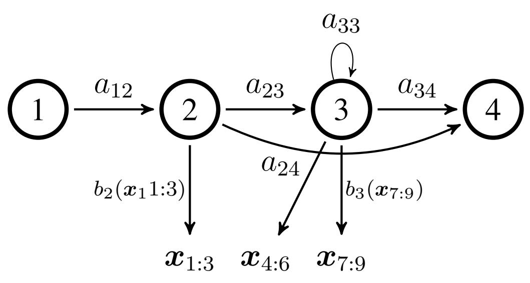
An approach, related to ANN-HMM hybrid modelling is the tandem approach. In a tandem system, first, an ANN is trained similarly to a hybrid system (usually as in Section 3.9.2). Then the output of one of the layers is used as the input vectors to a GMM-HMM system. The output of the output layer was originally used as a feature modelled by GMMs (Hermansky et al., 2000), although more recently the output of a hidden layer is used (Fousek et al., 2008; Park et al., 2011; Wang et al., 2015; Zhang and Woodland, 2017). This hidden layer usually has a very small width in comparison to the preceding and succeeding layers, which is why it is called a bottleneck layer (Grezl et al., 2007).

3.3.5 Low Frame Rate HMM

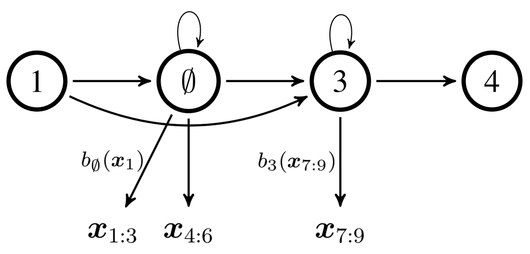
The constituent HMMs used in the AM were introduced as having three emitting states and two non-emitting states. Figure 3.5a shows an alternative HMM topology which contains only one emitting state. Also, the HMM operates at a three times lower frequency, meaning that one HMM state models three acoustic vectors at full frame rate. This is also referred to as low frame rate acoustic modelling (Pundak and Sainath, 2016). These low frame rate acoustic models are computationally more efficient, however, the relationship between acoustic vector



(a) Single state HMM topology.



(b) Two state "Chain" topology with skip connection used in LF-MMI.



(c) CTC-style HMM topology. The emission parameters associated with $\emptyset$ are shared between all constituent HMMs. In a CTC model, if the same constituent HMM occurs twice in a row, the blank-state $\emptyset$ cannot be skipped.

Figure 3.5 Low frame-rate HMM topologies, where three output states emit nine acoustic vectors $\mathbf{x}_{1:9}$. a_{ij} is the probability of transitioning to state j when being in state i. $b_i(\mathbf{x}_t)$ is the probability of emitting vector $\mathbf{x}_t$ when being in state i. a_{ij} is a discrete probability, while $b_i(\mathbf{x}_t)$ continuous probability. The CTC-model ignores transition probabilities a_{ij}.

and state is now more complex. Therefore, these low frame rate AMs are exclusively used with ANNs and not with GMMs. To reduce the frequency by a factor of three, there are multiple options. The vectors can simply be sub-sampled ($\mathbf{x}_t^{\text{lf}} = \mathbf{x}_{3t}$) or concatenated ($\mathbf{x}_t^{\text{lf}} = \mathbf{x}_{3t-2:3t}$). Also, the ANN itself can be used to reduce the frame rate. For example, if only every third output of the TDNN in Figure 2.3 is used, that would constitute a reduction in frequency of a factor of 3 while also using all acoustic vectors. Also, multi-layered RNNs can be used where one of the RNN-layers only takes an input every three (input) time-steps. RNNs and TDNNs can also be combined (Peddinti et al., 2017). Figure 3.5b shows a HMM-topology that is in HMMs trained using Lattice-Free Maximum Mutual Information (LF-MMI) (Povey et al., 2016) (see Section 3.9.3.1).

3.3.6 Connectionist Temporal Classification

Another HMM-topology is used in the Connectionist Temporal Classification (CTC) approach (Graves et al., 2006). It is illustrated in Figure 3.5c. Note that in the CTC-topology when the same constituent HMM occurs in succession, the blank symbol is not skippable. This results in the benefit that the output sequence can be directly obtained from the state sequence by removing repeated symbols *i.e.*:

$$\mathcal{B}(a\emptyset\emptyset abb\emptyset) = \mathcal{B}(\emptyset aa\emptyset a\emptyset b) = aab \tag{3.33}$$

where $\mathcal{B}$ is the function that produces that transcription sequence from the state sequence. The CTC model also ignores the transition probabilities a_{ij}. Effectively, any a_{ij} represented by an arrow is set to 0.5. In LF-MMI models, transition probabilities are also sometimes ignored (Hadian et al., 2018). In CTC, the state priors $P(s = i)$ are often ignored. Given that the blank-state $\emptyset$ prior is much higher than for the other symbols, the posterior probability of the blank-state $\emptyset$ is often penalised by an empirical value (Sak et al., 2015a,b). Beyond using a different HMM-topology, CTC models are also trained using a conditional maximum likelihood criterion that operates at the sequence level. This training criterion will be explained in Section 3.9.3.2.

3.4 Language Modelling

As described in Equation (3.4) a solution to ASR can be described as combining an acoustic model $p(\mathbf{X}|\mathbf{w})$ with a Language Model (LM) $P(\mathbf{w})$, where the AM is usually a HMM as described above. The task of the LM is to provide a prior probability for any sequence of words,

$$P(\mathbf{w}) = \prod_{m=1}^{M+1} P(w_m|w_{m-1}, w_{m-2}..., w_0) \tag{3.34}$$

where w_0 is the start of sentence symbol `<SOS>` and w_{M+1} is the end of sentence symbol `<EOS>`. LMs play an important role in many applications beyond ASR such as machine translation (Gulcehre et al., 2017; Koehn, 2009) or predictive text. In ASR, LMs are imperative for correct speech modelling because they put (soft) constraints on the possible word sequence. This is especially important in languages with a large number of homophones in which case an acoustic model by itself is not able to decide which word would correspond to the acoustic input.

In general, LMs operate at the word/token level and are used to provide an estimate for the probability of $P(w_m|w_{m-1}, w_{m-2}..., w_0)$ and then Equation (3.34) is used to obtain $P(\mathbf{w})$. For even a moderate vocabulary size, the number of different probabilities to be estimated for this model is prohibitively large leading to both storage and data sparsity problems. Hence, the history $(w_{m-1}, w_{m-2}..., w_0)$ needs to be summarised. Two methods will be presented. One is the N-gram model (Jelinek, 1991; Manning and Schütze, 1999), which truncates the history. The second will be using a neural-network to summarise the history.

3.4.1 N-Gram Language Models

In N-gram language models the history is truncated to the last $N-1$ tokens *i.e.* $(w_{m-1}, w_{m-2} \dots w_{m-N+1})$. The Maximum Likelihood (ML) estimate for N-gram models is given by:

$$P(w_m|w_{m-1}, w_{m-2}..., w_{m-N+1}) = \frac{\text{count}\,(w_m, w_{m-1}..., w_{m-N+1}; \mathcal{D}_{\text{text}})}{\text{count}\,(w_{m-1}, w_{m-2}..., w_{m-N+1}; \mathcal{D}_{\text{text}})} \tag{3.35}$$

where count $(w_l, w_{l-1}..., w_{l-N+1}; \mathcal{D}_{\text{text}})$ is the number of occurrences of $[w_l, w_{l-1}..., w_{l-N+1}]$ in the text-training corpus $\mathcal{D}_{\text{text}}$. For training the LM, a larger corpus of text can be used that does not need to have paired audio-data.

For a vocabulary sized $||V||$, $P(w_m|w_{m-1}, w_{m-2}..., w_{m-N+1})$ needs to contain values for $||V||^N$ probabilities. Within any text corpus, the N-gram statistics will follow Zipf's law (inverse power law) (Ferrer-i Cancho and Sole, 2002). This means that, when N is large, almost all N-gram units are unseen and others do not have sufficient occurrences in the training corpus for reliable estimation giving rise to a severe data sparseness issue. Unseen N-gram units will have a probability of zero and thus prohibit the ASR system from generating outputs containing them, regardless of how unambiguous the acoustic signal is. This can be solved using multiple methods. A simple method is to add a count of 1 to each count, which causes all N-grams to be possible. This is equivalent to using a Dirichlet prior on the N-gram probabilities. In practice, the estimation of infrequent N-grams is done by "backing-off" to N-gram models with shorter histories (Katz, 1987; Kneser and Ney, 1995).

3.4.2 Neural Network Language Models

There are two main issues associated with N-gram language models. The first is data sparsity, which as mentioned above can be solved using sophisticated "smoothing" techniques that use a smaller value for N in cases where not enough data was available in training to estimate specific N-gram probability. The second issue is the Markov assumption *i.e.* that the probability of a word is only dependent on the preceding $N-1$ words. Neural network LMs, in particular

RNN LMs, can be used to solve these issues. An RNN LM summarises the history as

$$\boldsymbol{h}_m = \varphi((w_{m-1}, w_{m-2}..., m_0)) = \varphi(w_{m-1}, \boldsymbol{h}_{m-1}) \tag{3.36}$$

where as described in Section 2.1.5. In this way, the entire history is being used and the RNN learns which information from the entire history to include and which to discard. RNN LMs use recurrent connections between hidden layers to model the history of words rather than truncating the history to a rather arbitrary length of words. They have become increasingly popular in recent years and have shown impressive results in a range of tasks and systems (Mikolov et al., 2010).

The RNN LM consists of an RNN as described in Section 2.1.5. In order to use it for language modelling the input at each time-step is a one-hot encoding of the previous word,

$$[\mathbf{x}_m]_i = \begin{cases} 1 & i = w_{m-1} \\ 0 & i \neq w_{m-1} \end{cases} \tag{3.37}$$

where the length of $\mathbf{x}_m$ is the size of the vocabulary. These vectors can be fed into a RNN. The RNN will summarise the previous words into a final hidden vector: $\boldsymbol{h}_m^{(L-1)}$. Based on $\boldsymbol{h}_l^{(L-1)}$ the probability distribution $P(w_m|w_{m-1}, w_{m-2}..., w_0)$ is obtained by passing $\boldsymbol{h}_l^{(L-1)}$ through an affine transformation followed by the Softmax activation function (Equation (2.4)),

$$P(w_m = i|w_{m-1}, w_{m-2}..., w_0) \approx P(w_m|\boldsymbol{h}_m^{(L-1)}) \tag{3.38}$$

$$\approx \frac{\exp\left(\boldsymbol{h}_m^{(L-1)T}\boldsymbol{e}_i + b_i\right)}{\sum_{i'=1}^{\|V\|} \exp\left(\boldsymbol{h}_m^{(L-1)T}\boldsymbol{e}_{i'} + b_{i'}\right)} \tag{3.39}$$

where $\boldsymbol{e}_i$ are the rows of the weight-matrix of the output layer and $\|V\|$ is the size of the vocabulary. In many RNNs the output-matrix and the first matrix (that is multiplied with $\mathbf{x}_m$) are tied (Inan et al., 2017; Press and Wolf, 2017).

Recently, language models that use the Transformer (see Section 2.1.7) architecture have become popular (Dai et al., 2019; Irie et al., 2019; Sun et al., 2021).

3.5 Decoding and Alignment

After training the LM $P(\mathbf{w})$ and the AM $p(\mathbf{X}|\mathbf{w})$ the missing piece is how to solve Equation (3.3).

$$\begin{aligned} \mathbf{w}^\star &= \arg\max_{\mathbf{w}} \quad P(\mathbf{w}|\mathbf{X}) && (3.3) \\ &= \arg\max_{\mathbf{w}} \quad P(\mathbf{w})p(\mathbf{X}|\mathbf{w}) && (3.40) \\ &= \arg\max_{\mathbf{w}} \quad \log P(\mathbf{w}) + \log p(\mathbf{X}|\mathbf{w}) && (3.41) \end{aligned}$$

The process denoted $\arg\max_{\mathbf{w}}$ is called *decoding*. The decoding process relies on creating a computational graph that combines the AM and the LM. It also relies on combining the parameters of the physical HMM states with the decision tree used to cluster context-dependent states and with the pronunciation dictionary. This graph can be constructed statically (Mohri et al., 2002, 2008; Young et al., 1989), dynamically (Odell et al., 1994) or in a manner that dynamically adds LM information to a static graph (Demuynck et al., 2000; Odell, 1999). The decoding task corresponds to finding the most probable path through the graph.

3.5.1 Viterbi Decoding

Before describing how decoding is done for Equation (3.3) in Section 3.5.2, this section will describe how to find the ML state-sequence through the composite HMM is found *i.e.*:

$$\mathbf{s}^\star = \arg\max_{\mathbf{s}} p(\mathbf{X}, \mathbf{s}|\mathbf{q}) \tag{3.42}$$

This is also the state-sequence used for frame-level cross-entropy training of ANN-HMM models in Section 3.9.2 and to estimate the prior state probabilities that are used in Equation (3.26).

Equation (3.42) can be solved using *Viterbi decoding*. Viterbi decoding is a dynamic programming algorithm for finding the most likely sequence of hidden states for a given HMM. Viterbi decoding requires the HMM to be known in advance, *i.e.* the sequence of sub-word units needs to be known in order to build the composite HMM from the smaller constituent HMMs. Similar to the forward-backward algorithm, Viterbi decoding exploits the independence assumptions of the HMM to reduce the algorithmic complexity from $\mathcal{O}(S^T)$ down to $\mathcal{O}(ST)$. Where S is the number of logical states in the HMM.

Let $\psi_j(t)$ represent the likelihood of the ML state-sequence the ends in state $s_t = j$ *i.e.* after observing $\mathbf{x}_{1:t}$.

$$\psi_j(t) = \max_{\mathbf{s}} p(\mathbf{x}_{1:t}, s_{1:t}|\mathbf{q}) \tag{3.43}$$

The log-probability $\log p(\mathbf{x}_{1:t}, s_{1:t}|\mathbf{q})$ can be decomposed into

$$\begin{aligned}\log p(\mathbf{x}_{1:t}, s_{1:t}|\mathbf{q}) = &\log p(\mathbf{x}_{1:t-1}, s_{1:t-1}|\mathbf{q}) \\ &+ \log P(s_t|\cancel{\mathbf{x}_{1:t-1}}, \cancel{s_{1:t-2}}, s_{t-1}, \mathbf{q}) \\ &+ \log p(\mathbf{x}_t|s_t, \cancel{\mathbf{x}_{1:t-1}}, \cancel{s_{1:t-1}}, \mathbf{q})\end{aligned} \tag{3.44}$$

where again the strikeouts $\cancel{\mathbf{x}_{1:(t-1)}}$ are based on the independence assumptions of the HMM. Therefore $\psi_j(t)$ can be recursively computed using,

$$\psi_j(t) = \max_i\{\psi_i(t) + \log P(s_t = j|s_{t-1} = i)\} + \log p(\mathbf{x}_t|s_t = j) \tag{3.45}$$

When computing each $\psi_j(t)$, the maximum operation removes those paths that cannot form the overall optimal path. It is this maximum operation that causes the algorithm to result in a linear time-complexity rather than an exponential time complexity. In the case of an HMM that ends with non-emitting state $s_{T+1} = S$ the algorithm terminates with,

$$\psi_N(T+1) = \max_i\{\psi_i(t) + \log P(s_t = N|s_{t-1} = i)\} \tag{3.46}$$

A direct computation of the likelihood would usually lead to numerical underflow which is why the logarithm is used.

3.5.2 Large Vocabulary decoding using the Token Passing Model

The above algorithm obtains the ML state-sequence $\mathbf{s}^\star$ only for a fixed HMM $\mathbf{q}$. However, the real objective of decoding is to find the path associated with the maximum value of $P(\mathbf{w}|\mathbf{X})$, in which case the LM probabilities have to be taken into account. Finding the value of $P(\mathbf{w}|\mathbf{X})$ for a specific $\mathbf{w}$ is still straightforward. When transitioning from one constituent HMM to another which belongs to another word, say $w_m \rightarrow w_{m+1}$, then the LM probabilities can be included in the calculation of $\log P(s_t = j|s_{t-1} = i)$ used in Equation (3.45). $\log P(s_t = j|s_{t-1} = i)$ is then the sum of the log transition probability from state $s_{t-1} = i$ to the non-emitting end state of the first HMM and the log LM probability $\log P(w_{m+1}|w_m)$. The log LM probability is usually linearly scaled by a grammar scaling factor κ (Bahl et al., 1980). This is necessary since HMM acoustic models often produce a wider dynamic range of likelihood values due to the underestimation of the likelihood arising from invalid assumptions (Woodland and Povey, 2002). Effectively, the independence assumptions of the HMM are too strong. Therefore, the AM probabilities become too sharp (low entropy). To compensate for this the LM probabilities are made sharper by the multiplication with κ.

In HTK (Young et al., 2015) the implementation of the Viterbi algorithm uses the *Token Passing Model*. In this conceptual model, at time $t-1$, a HMM state i holds a movable token that contains $\phi_i(t-1)$ and the state-sequence to achieve it. At time t, a copy of every token in every state i is passed to all of the possible succeeding states j, and each token has its value increased by $[\log P(s_t = j|s_{t-1} = i) + \log p(\mathbf{x}_t|s_t = j)]$. Then for each state, all tokens except for the one with the highest value are discarded as evident in Equation (3.45). When the sub-word unit HMM of different words are connected together then, as described above, the token value is further increased by $\log P(w_m \rightarrow w_{m+1})$ in the case of a change of word.

If $\log P(w_m \rightarrow w_{m+1})$ is a bigram LM, then no special care needs to be taken. However, if an N-gram LM is used with $N > 2$, then the search graph needs to be extended to include context-dependent copies of the words. An utterance is assumed to end with a silence which thus defines the endstate S needed for the Viterbi algorithm to terminate (see Equation (3.46)). When used for word-level decoding rather than pure Viterbi decoding, the Token Passing Algorithm will simply keep the word history and not the exact state history. The token passing model is also useful for finding N-best paths (see Section 3.7.1). Each HMM state is allowed to hold multiple tokens at the same time, each of which will carry a different word history. Tokens from different preceding words are regarded as distinct.

The token passing algorithm on its own can already find the most probable path $\mathbf{w}^\star$ for the input utterance $\mathbf{x}$. However, this process is prohibitively expensive due to the large search space. Therefore approximations have to be done in order to find the most probable word sequence. The Viterbi search is time-synchronous thus reducing the computation by reducing the width of the search space. At each time step, any token whose value is smaller than a certain threshold defined relative to the maximum among the values of all tokens is deactivated. Therefore, the search space is constrained to a certain beam, defined by the difference in log probability (the beam width). Therefore this kind of search is called *beam search*. Complex decoding algorithms use a range of further pruning methods. The pruning can prune the optimal path through the computational graph, which causes pruning errors or search errors. Therefore a trade-off between accuracy and computation time has to be found. Beyond the aforementioned grammar scaling factor κ, a negative-valued word insertion penalty ι is used to avoid word insertions (see Section 3.6). This turns Equation (3.41) into

$$\mathbf{w}^\star = \arg\max_{\mathbf{w}} \kappa \log P(\mathbf{w}) + \log p(\mathbf{X}|\mathbf{w}) + \iota||\mathbf{w}|| \tag{3.47}$$

3.5.3 Decoding using WFSTs

Decoding can also be done based on the Weighted Finite State Transducer (WFST) framework (Mohri et al., 2002, 2008). The WFST framework allows the search graph to be computed in advance and then to be used for many utterances.

A WFST is a graph where each transition between nodes is associated with a pair of symbols and a cost of the transition. The pair of symbols are input and output symbols. Therefore, the WFST relates pairs of symbol sequences to each other. It essentially transforms an input sequence into (possibly many) output sequences and their associated costs. The great advantage of WFSTs is that two WFSTs can easily be combined (called composition). For example, the LM G can be represented as a WFST where the input and output pairs are words and the cost is the negated log-probability of the LM. The pronunciation dictionary L will have as its input sequence a phone sequence and as its output sequence the word sequence. The context dependency C (*e.g.* biphones or triphones) maps from context-dependent sub-word units to context-independent sub-word units. The HMM-topology H maps from HMM-states to context-dependent sub-word units. H will also encode state-tying from Section 3.3.3 and the transition probabilities a_{ij}. The individual WFSTs H, C, L and G can be combined to one WFST written as $H \circ C \circ L \circ G$. The WFST $C \circ L \circ G$ maps from context-dependent sub-word units to words. The WFST $H \circ C \circ L \circ G$ maps from HMM-states to words. Besides composition, two other operations are important for WFSTs: determinisation and minimisation. In a deterministic automaton, each state has at most one transition for any given input label. This means that the WFST contains at most one path matching any given input string. This reduces the time and space needed to process the input sequence. Determinisation is the process of making a WFST deterministic. Even if L and G are deterministic that does not imply that $L \circ G$ is deterministic. The way in which the pronunciation dictionary L is normally represented *i.e.* as a dictionary mapping a word to a pronunciation is not deterministic. Using a tree-organised pronunciation dictionary would be deterministic. Minimisation transforms the WFST into an equivalent transducer that has the fewest number of states and transitions. Determinisation and minimisation need to be applied to the composite WFST to make it efficient to use. This results in:

$$HCLG = min(det(H \circ C \circ L \circ G)) \tag{3.48}$$

$HCLG$ now has one start state, each state has a final cost and there is a set of arcs between the states, where each arc has an input label, an output label, and a weight. In $HCLG$, the input tokens are the identifiers of the clustered physical HMM states (effectively encoding an ID for the pdf $b_i(t)(\mathbf{x})$, which is the only part missing from this graph), and the output tokens represent words. For both the input and output tokens, the special symbol ϵ exists as a NULL-token. The important part to remember here is that $HCLG$ can be created ahead of decoding. Dynamic graph construction cannot do this and hence also cannot take advantage of the improved decoding speed due to minimisation and determinisation. Viterbi decoding with beam pruning (similar to Section 3.5.2) can be applied to $HCLG$.

Some details are missing from this description of WFSTs for ASR such as the fact that some disambiguation symbols need to be included in L such that no two words can have the same pronunciation (which isn't true for "red" and "read"). In the final graph the disambiguation symbols can be replaced by ϵ.

3.6 Evaluation

Ref	***	paid	off	in	a	(y-)	year	(HES)	or	six	months
Hyp	pay	it	off	in	a		year	**	**	six	months
Type	I	S	C	C	C		C		D	C	C

Table 3.2 Levenshtein distance calculation. Error types are Correct (C), Insertion (I), Deletion (D) and Substitution (S). Here, partial words (*e.g.* (y-)) and hesitations (HES) can be optionally deleted.

The most commonly used metric in evaluating the performance of ASR systems is the Word Error Rate (WER). This measure calculates the ratio of words which differ between the hypothesised word sequence that is generated by the ASR system and the corresponding reference transcription. However, these transcriptions do not necessarily have the same length. Therefore, one cannot simply compare the two words in the two transcriptions of the same index. The sequences must therefore firstly be aligned using a dynamic programming procedure. The dynamic programming algorithm will find the alignment between the two sequences that minimises the *Levenshtein* distance between them. The Levenshtein distance is defined as a weighted sum of the following error types:

- **Substitution errors**, which represent the situation where a reference word is aligned with a *different* hypothesised word. This means a word is misrecognised as another.
- **Deletion errors**, which represent the situation where a reference word is not aligned with *any* word in the hypothesis. This indicates that a word has been omitted by the ASR system.

- **Insertion errors**, which represent the situation where an *additional word* is present in the hypothesis that cannot be aligned with a word in the reference. This indicates an additional word has been produced by the ASR system.

The WER is then calculated as the number of errors divided by the number of words in the reference *i.e.*,

$$\text{WER} = 100\% \cdot \frac{\text{S} + \text{D} + \text{I}}{\text{M}} = 100\% - 100\% \cdot \frac{\text{C} - \text{I}}{\text{M}} \tag{3.49}$$

where S, D, I, C, M are the number of substitutions, deletions, insertions, correct words and words in the reference, respectively. This means that technically the WER can be higher than 100%. Table 3.2 gives an illustration of the alignment and subsequent errors of an utterance.

This form of WER metric only considers the 1-best transcription produced by an ASR system. If the output of the ASR system is a lattice or an n-best list, the oracle error rate is used. This is the lowest possible WER of any sequence in the lattice or n-best list.

3.7 Hypothesis Spaces

The decoding procedure in Section 3.5.2 is rather expensive. It becomes even more expensive when using N-gram language models with $N > 2$, because they significantly increase the number of nodes in the decoding procedure. Decoding would be faster if the number of possible hypotheses is constrained. Within this constrained *hypothesis space* (in particular N-best lists Section 3.7.1 and lattices Section 3.7.2) a higher-order language model can be applied. For the RNN LMs of Section 3.4.2 these constraint hypothesis spaces are indispensable. Not only the LM can be exchanged, but also the AM can be exchanged and the utterance can be re-decoded much faster. Using a different LM or AM for an existing N-best list or lattice is called *rescoring*. Decoding first using an N-gram model with a small N (*e.g.* $N = 2$) and then decoding a constraint hypothesis space that was rescored with an N-gram model with a larger N (*e.g.* $N = 4$) is called *multi-pass* decoding (Young, 1996). An N-Best list can also be used as the input to a downstream application such as a spoken dialogue system.

3.7.1 N-Best Lists

An example for an N-Best list is given in Table 3.3. The N most probable hypothesis are listed in the N-Best list. An N-Best list is often used for LM-rescoring particularly for RNN-LM-rescoring. The word sequence is already given and hence computing a new value for $P(\text{w})$ is easy.

#	$\log p(\mathbf{X}\|\mathbf{w})$	$\log P(\mathbf{w})$	$\kappa \log P(\mathbf{w}) + \log p(\mathbf{X}\|\mathbf{w})$	hypothesis
1	-16707.6	-6.5	-16785	<s> you know </s>
2	-16717.0	-9.21	-16828	<s> uh you know </s>
3	-16798.3	-2.8	-16832	<s> yeah </s>
4	-16716.4	-10.1	-16837	<s> um you know </s>
5	-16773.1	-5.8	-16843	<s> and uh </s>
6	-16723.6	-10.1	-16845	<s> you know um </s>

Table 3.3 NBest-List for the utterance “you know”. With grammar scale factor $\kappa = 12$.

3.7.2 Lattices

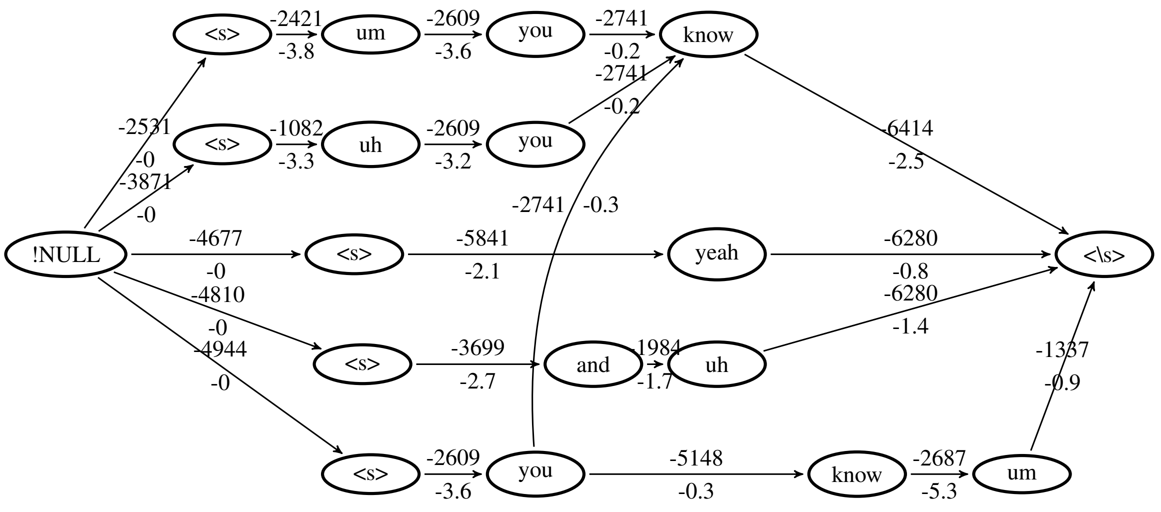


Figure 3.6 A (heavily pruned) decoding lattice. Each node represents a word and the time of the end of the word. Each arc is associated with the acoustic log-likelihood (above) and the unscaled language model log-probability (below). For the acoustic likelihood and the language model probability the log-scale is used. The x-axis represents the time from 0.00s to 1.26s.

A lattice is a compact representation of multiple overlapping hypotheses. It represents a large number of hypotheses much more efficiently than an N-best list. The lattice takes the form of an acyclic graph structure where nodes represent the ends of words at particular points in time. This means that there will be multiple nodes for the same word, which represent the same word but at slight shifts in time. A number of arcs then connect these nodes. The arcs thus define which paths through the lattice are feasible hypotheses. The lattice is pruned in such a way that it contains as few nodes and arcs as possible whilst still representing all hypotheses that have a large probability $P\left(\mathbf{w}|\mathbf{X}\right)$ associated with them. The arcs of the lattice also contain the acoustic likelihood of the maximum likelihood path through the constituent HMMs of the word corresponding to the node at the end of the arc, for the time duration defined by two nodes of the arc. The arc also contains the probability of the word transitioning from one node to the next. Lattices are used for rescoring with larger N-gram models. For decoding using a 4-gram LM, a bigram LM can be applied in the first decoding pass, while in the second pass the 4-gram LM is used to rescore the word hypotheses of the lattice by keeping the acoustic model scores produced in the first pass. To do so, however, first the original lattice needs to be expanded to include context-dependent copies of the words. This means that words with different histories need to have separate nodes.

A lattice is shown in Figure 3.6. The decoded utterance is the same as in Table 3.3. Given that certain nodes are shared between utterances, a lattice can store a large N-best list much more efficiently.

3.7.3 Confusion Networks

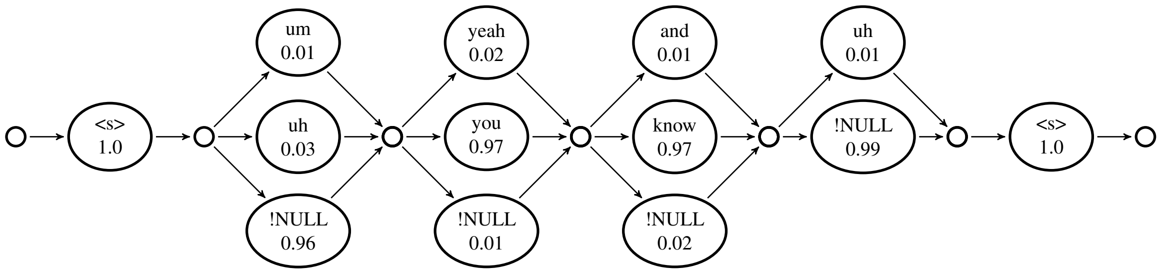


Figure 3.7 An example of a confusion network. The small circles are the nodes. The large circles with text are the labels of the arcs.

An alternative dense representation of the most likely hypotheses are *confusion networks* (Evermann and Woodland, 2000; Mangu et al., 2000). A confusion network is a directed acyclic graph similar to a lattice. However, all the outgoing arcs of one node are the incoming arcs for the next node. The nodes correspond to timestamps only, and the arcs correspond to words (which can be NULL-words). Furthermore, the arcs store the probability of each word transition (or skip-transition) normalised over the arcs connecting two nodes. This means that these probabilities are word-posterior probabilities $P(w_m|\mathbf{X})$. Obtaining word-posterior probabilities and then finding the hypothesis with the lowest sum of word-posterior probabilities is one of the motivations for confusion networks.

A confusion network is shown in Figure 3.7. The decoded utterance is the same as in Table 3.3 and Figure 3.6.

3.7.3.1 Posterior Probability Decoding

Usually, the ASR system is decoded based on Equation (3.3).

$$\mathbf{w}^\star = \arg\max_{\mathbf{w}} P(\mathbf{w}|\mathbf{X}) \tag{3.3}$$

Equation (3.3) minimises the sentence error rate (the probability of having at least one error in the hypothesis). The more commonly used metric for ASR systems is, however, the WER of Section 3.6. Then if we view $\mathbf{w}^\star$, not as the word sequence but the "aligned" sequence "Hyp" in Table 3.2 (*i.e.* including possible 'NULL' words), finding the minimum WER hypothesis is equivalent to finding the hypothesis with the maximum number of correct words, *i.e.*

$$\begin{aligned}
\mathbf{w}^\star &= \arg\max_{\mathbf{w}} \mathbb{E}[\text{words correct}(\mathbf{w})|\mathbf{X}] && (3.50)\\
&= \arg\max_{\mathbf{w}} \sum_m \mathbb{E}[\text{correct}(w_m)|\mathbf{X}] && (3.51)\\
&= \arg\max_{\mathbf{w}} \sum_m P(w_m|\mathbf{X}) && (3.52)
\end{aligned}$$

Hence, finding the minimum WER hypothesis is equivalent to finding the hypothesis with the largest sum of word-posterior probabilities. This form of decoding is easy in confusion networks as within each segment of the confusion network, the word (or !NULL) is chosen with the highest associated probability. This posterior probability decoding is a form of Minimum Bayes Risk (MBR) decoding. N-Best (Goel and Byrne, 2000) and Lattice (Xu et al., 2011) MBR decoding are alternatives to confusion network decoding. The speech recognition experiments in Chapter 7 and Chapter 8 both use confusion network decoding.

3.7.3.2 Confusion Network Creation

The confusion network is built by starting from a lattice. Arcs of the lattice are step-by-step grouped together until the confusion network structure is built. Whenever arcs are grouped together, their probabilities are added. This probability can be calculated using the forward-backward algorithm. The forward-backward algorithm is applied on the node level instead of the state level. The arc probability is then equivalent to ξ in Section 3.3.2. Usually, arcs are grouped together in two ways. First, arcs that end in the same word and have large amounts of time overlap are merged. Once, this has been done and the topology of the lattice is similar to that of a confusion network, words with similar pronunciations are merged. Finally, the !NULL-nodes are added, and the probabilities of each word position are normalised.

3.8 Confidence Estimation

In confidence estimation, the ASR system is supposed to give an indication of the reliability of the given transcription. Confidence estimation ascribes confidence scores to the output units and can be done at different levels of granularity (phone, word, utterance). Confidence scores are crucial in many tasks that rely on the output of an ASR system. These include out-of-vocabulary detection (Young, 1994), keyword spotting (Wilpon et al., 1990) and Spoken Dialogue Systems (SDSs) (Young et al., 2013).

By taking advantage of the statistical nature of ASR systems, the posterior probabilities of the hypotheses can be considered to be a reflection of the true accuracy of the output transcription. As discussed in Section 3.5.2 the ASR system tries to find the sequence with the highest posterior probability. Intuitively, multiple competing hypotheses with similar posterior probabilities should be an indication of high uncertainty. In turn, if the posterior probability of the most probable sequence is significantly higher than that of other corresponding hypotheses, it means that the ASR system is confident.

As described in Section 3.7, competing hypotheses of the ASR system can be efficiently represented using N-best lists, lattices or confusion networks. The information present in these structures can be used as features in a system that tries to compute the confidence scores. In a confusion network the arc probabilities directly correspond to confidence scores on the word level.

The confidence scores of confusion networks are reasonably good relative to each other, but often poorly represent the actual probability of a word being correct. In order to improve these scores a piece-wise linear mappings form the arc probability to the confidence score can be trained. Furthermore, RNNs can be used to directly predict the confidence score (Kalgaonkar et al., 2015; Kastanos et al., 2020; Ragni et al., 2018). An RNN that is time-synchronous with the output sequence, predicts the probability of each word being correct and can be trained on a corpus. The input features to the RNN can include the duration of the word, the confidence estimated by the confusion network, the decoded word itself, the acoustic and language model probabilities from the corresponding arc of the lattice as well as the order of the N-gram used for the language model probability (after backoff).

3.9 Acoustic Model Training

This section describes estimating the parameters of the AM. These are the transition probabilities a_{ij} and the emission probabilities $b_i(t)$. As in Chapter 2, if it's not necessary to specify further, the parameters will be represented by θ. In supervised learning, the ASR system will be trained based on a training set $\mathcal{D}_l = \{\mathbf{X}_n, \hat{\mathbf{w}}_n\}_{n=1}^{N}$.

3.9.1 Generative Maximum Likelihood

Similar to ANNs (Equation (2.26)), ML can be used to train an ASR system. The objective is to maximise the probability that the trained model assigns to the training set $\mathcal{D}_l$.

$$\theta^{ML} = \underset{\theta}{\text{argmax}} \prod_{n=1}^{N} P_\theta(\hat{\mathbf{w}}_n, \mathbf{X}_n) = \underset{\theta}{\text{argmax}} \sum_{n=1}^{N} \log P_\theta(\hat{\mathbf{w}}_n, \mathbf{X}_n) \tag{3.53}$$

When the AM is trained in a generative fashion on its own (*i.e.* with a fixed language model), the word sequence is treated as fixed and the model is trained to maximise the probability of the acoustic vectors $\mathbf{X}_n$,

$$\theta^{ML} = \underset{\theta}{\text{argmax}} \sum_{n=1}^{N} \log P_{\theta'_{\text{LM}}}(\hat{\mathbf{w}}_n) p_\theta(\mathbf{X}_n|\hat{\mathbf{w}}_n) \tag{3.54}$$

$$= \underset{\theta}{\text{argmax}} \sum_{n=1}^{N} \log p_\theta(\mathbf{X}_n|\hat{\mathbf{w}}_n) \tag{3.55}$$

For simplicity, in the following parts of this Section, the utterance index n will be omitted and the optimisation will be derived for a single utterance.

3.9.1.1 Expectation-Maximisation

For HMM-GMM models, Equation (3.55) is usually optimised using Expectation Maximisation (EM). EM is an iterative optimisation method that can be used for many latent variable models. In the case of an AM, the state-sequence $\boldsymbol{s}$ through the composite HMM is the latent variable. In EM, Equation (3.55) is not directly optimised, but instead an auxiliary function $\mathcal{F}$ is optimised[3], where

$$\mathcal{F}(f(\boldsymbol{s}), \theta) = \sum_{\boldsymbol{s}} f(\boldsymbol{s}) \log\left(p(\mathbf{X}, \boldsymbol{s}|\hat{\mathbf{w}})\right) - \sum_{\boldsymbol{s}} f(\boldsymbol{s}) \log\left(f(\boldsymbol{s})\right) \tag{3.56}$$

$$= \log p(\mathbf{X}_n|\hat{\mathbf{w}}_n, \theta) - \sum_{\boldsymbol{s}} f(\boldsymbol{s}) \log\left(\frac{f(\boldsymbol{s})}{P(\boldsymbol{s}|\mathbf{X}, \hat{\mathbf{w}})}\right) \tag{3.57}$$

where $f(\boldsymbol{s})$ is any valid probability distribution with respect to (w.r.t) $\boldsymbol{s}$. Each iteration of the EM algorithm consists of two steps. The first step is the E-step, in which the posterior distribution over the latent variables ($P(\boldsymbol{s}|\mathbf{X}, \hat{\mathbf{w}})$) is found, and $f(\boldsymbol{s})$ is set to $P(\boldsymbol{s}|\mathbf{X}, \hat{\mathbf{w}})$. This

[3]Equation (3.55) will have a sum over all s $\in \mathcal{S}$ within the $\log(\cdot)$, hence there is no closed form solution

sets the second term in Equation (3.57) to zero, which is the KL-divergence between $f(\boldsymbol{s})$ and $P(\boldsymbol{s}|\mathbf{X}, \hat{\mathbf{w}})$. In a HMM, finding $P(\boldsymbol{s}|\mathbf{X}, \hat{\mathbf{w}})$ consists of finding $\gamma_i(t)$ (Equation (3.22)) and $\xi_{ij}(t)$ (Equation (3.23)).

The second step, the M-step, consists of maximising $\mathcal{F}$ w.r.t. θ. Here, Equation (3.56), is more useful as the M-step now consists of maximising the expectation of $p(\mathbf{X}, \boldsymbol{s}|\hat{\mathbf{w}})$ w.r.t. the distribution $f(\boldsymbol{s})$. The EM algorithm monotonically increases $\mathcal{F}$ in each iteration because both the E- and the M-step increase $\mathcal{F}$. Since after each E-step $\mathcal{F}(f(\boldsymbol{s}), \theta) = \log p_\theta(\mathbf{X}_n|\hat{\mathbf{w}}_n)$, the EM algorithm also increases the likelihood. Technically, the likelihood either increases or stays constant, but never decreases.

The update equation (in the M-step) for the transition probabilities a_{ij} are,

$$a_{ij}^{new} = \frac{\sum_{t=0}^{T} \xi_{ij}(t)}{\sum_{t=0}^{T} \gamma_i(t)} \tag{3.58}$$

For the GMM-parameters there is an additional set of latent variables. A GMM models a distribution where a sample belongs to a class and each of these classes is distributed according to a Gaussian. Which class a current sample belongs to is the additional latent variable. Hence, we need to introduce another set of posteriors

$$\begin{aligned}
\gamma_{i,k}(t) &= P_\theta(s_t = i, c_t = k|\mathbf{X}, \mathbf{q}) && (3.59)\\
&= P_\theta(s_t = i|\mathbf{X}, \mathbf{q}) P_\theta(c_t = k|s_t = i, \mathbf{X}, \mathbf{q}) && (3.60)\\
&= \gamma_{i,k}(t) \frac{\pi_{ik} \mathcal{N}(\mathbf{x}_t; \boldsymbol{\mu_{ik}}, \Sigma_{ik})}{b_i(\mathbf{x}_t)} && (3.61)
\end{aligned}$$

Based on this state-mixture component posterior, the update equations are,

$$\begin{aligned}
\pi_{ik}^{new} &= \frac{\sum_t \gamma_{i,k}(t)}{\sum_t \sum_k \gamma_{i,k}(t)} && (3.62)\\
\boldsymbol{\mu_{ik}}^{new} &= \frac{\sum_t \gamma_{i,k}(t)\mathbf{x}_t}{\sum_t \sum_k \gamma_{i,k}(t)} && (3.63)\\
\Sigma_{ik}^{new} &= \frac{\sum_t \gamma_{i,k}(t)\mathbf{x}_t\mathbf{x}_t^T}{\sum_t \sum_k \gamma_{i,k}(t)} - \boldsymbol{\mu_{ik}}^{new} \boldsymbol{\mu_{ik}}^{newT} && (3.64)
\end{aligned}$$

It is important to stress that $\gamma_i(t)$, $\gamma_{i,k}(t)$ and $\xi_{ij}(t)$ were calculated using the old parameter values, after which $\gamma_i(t)$, $\gamma_{i,k}(t)$ and $\xi_{ij}(t)$ are treated as constants when deriving Equations 3.58 to 3.64. After setting $f(\boldsymbol{s})$ to $P(\boldsymbol{s}|\mathbf{X}, \hat{\mathbf{w}})$, they are part of the arbitrary distribution $f(\boldsymbol{s})$.

3.9.1.2 Gradient Descent

For ANN-HMM models, Equation (3.55) can be optimised using gradient-based methods (see Section 2.2.3). For a single sample the derivative w.r.t. the pre-softmax activation of the i-th output node at time-step t is:

$$\frac{\partial \log p_\theta(\mathbf{X}|\hat{\mathbf{w}})}{\partial z_{i,t}} = \gamma_k(t) - [\varphi(\mathbf{x}_t)]_i \tag{3.65}$$

where $z_{i,t}$ is the linear activation of node i in the output layer at time-step t and $[\varphi(\mathbf{x}_t)]_i$ is the i-th output of the ANN at time-step t. These gradients can then be used using a gradient *ascent* based optimiser. Comparing the gradient in Equation (3.65) to the gradients in Equation (2.31), it can be seen that this is equivalent to Cross Entropy (CE) training with labels $\hat{y}_i(t) = \gamma_i(t)$.

3.9.2 Frame-Level Classification

Though Equation (3.65) would make it possible to train ANN-HMM hybrid models, this is very rarely done, though it can be found in Senior (1994). Instead, the best possible state-sequence through the combined HMM is found using an initial model (*e.g.* an already trained GMM-HMM model) and the Viterbi algorithm of Section 3.5.1. Then the cross-entropy criterion (Equation (2.30)) can be used to train $p(s_t = i|\mathbf{x}_t) = y_i = [\varphi(\mathbf{x}_t)]_i$ where i is the HMM state at time t given by the maximum likelihood path from the Viterbi algorithm. Thus, the gradients used to optimise the ANN become:

$$\frac{\partial \mathcal{C}^{CE}}{\partial z_i(t)} = [\varphi(\mathbf{x}_t)]_i - \delta_{is_t^\star} \tag{3.66}$$

where $s_t^\star$ is the state at time t taken from the Viterbi algorithm and $\delta_{ii'}$ is the discrete Dirac delta function. This gradient would be used with a gradient *descent* based optimiser.

When training the ANN in this way, the transition probabilities are often kept as before, as they don't have a large influence on the overall behaviour of the model (Dahl et al., 2012). This is largely because the emission-probabilities calculated by the ANN have a much wider dynamic range than the transition probabilities.

3.9.3 Maximum Mutual Information

Instead of training the ASR system as a generative model as in Equation (3.55). We can train it as a discriminative model:

$$\theta^{ML} = \underset{\theta}{\operatorname{argmax}} \sum_{n=1}^{N} \log P_{\theta}(\hat{\mathbf{w}}_n|\mathbf{X}_n) \tag{3.67}$$

$$= \underset{\theta}{\operatorname{argmax}} \sum_{n=1}^{N} \log \frac{P_{\theta'_{\text{LM}}}(\hat{\mathbf{w}}_n)p_{\theta}(\mathbf{X}_n|\hat{\mathbf{w}}_n)}{p_{\theta}(\mathbf{X}_n)} \tag{3.68}$$

$$= \underset{\theta}{\operatorname{argmax}} \sum_{n=1}^{N} \log \frac{P(\hat{\mathbf{w}}_n|\theta'_{\text{LM}})p_{\theta}(\mathbf{X}_n|\hat{\mathbf{w}}_n)}{\sum_{i'} P_{\theta'_{\text{LM}}}(\mathbf{w}_{i'}|\theta'_{\text{LM}})p_{\theta}(\mathbf{X}_n|\mathbf{w}_{i'})} \tag{3.69}$$

where $\mathbf{w}_{i'}$ are all possible hypotheses for the input utterance $\mathbf{X}_n$.

The sum in the denominator would be prohibitively expensive to implement. However, this sum can be well enough approximated by summing only over those word sequences that contain most of the probability mass. Hence, a constraint hypothesis space called a lattice (see Section 3.7.2) is used. To obtain an efficient lattice the AM has to be pre-trained. Therefore, often the AM is first trained using ML or CE.

Section 3.5.2 introduced the grammar scaling factor κ. This should also be used in Maximum Mutual Information (MMI) training to compensate for the larger dynamic range of the AM scores. However, because the sum in the denominator is often dominated by a small number of hypotheses. Instead of applying the grammar scaling factor to the LM, the inverse of the grammar scaling factor ($\frac{1}{\kappa}$) is applied to the acoustic model (Woodland and Povey, 2002). This increases the number of confusable hypotheses within the denominator lattice. It also does not change the 1-best path. The resulting objective is,

$$\theta^{*} = \underset{\theta}{\operatorname{argmax}} \sum_{n=1}^{N} \log \frac{P_{\theta'_{\text{LM}}}(\hat{\mathbf{w}}_n)p_{\theta}(\mathbf{X}_n|\hat{\mathbf{w}}_n)^{1/\kappa}}{\sum_{i'} P_{\theta'_{\text{LM}}}(\mathbf{w}_{i'})p_{\theta}(\mathbf{X}_n|\mathbf{w}_{i'})^{1/\kappa}} \tag{3.70}$$

3.9.3.1 Lattice-Free MMI

Povey et al. (2016) introduced LF-MMI, which does not use a lattice and hence sums over all possible output sequences. To make this computationally feasible, a few changes need to be made. First, a low frame rate acoustic model (see Section 3.3.5) is used. Particularly the one in Figure 3.5a. The tree used to cluster states is derived from a GMM-HMM model. Second, rather than a word-level LM, a phone-level LM is used. Given that the number of phones is much smaller than the number of words, this significantly reduces the branching factor of the numerator graph. Typically 4-gram phone LM is used. However, at and below the trigram level, there is no smoothing or pruning. This means that there are many sequences whose probability is zero, which further reduces the size of the denominator graph. Thirdly, instead of training on entire utterances, training is done on shorter “chunks” (1.5 s). The appropriate numerator and denominator graphs are extracted using a GMM-HMM system.

3.9.3.2 CTC Training

The CTC approach to ASR was explained in Section 3.3.6. This section will explain the training. CTC models are trained with the conditional maximum likelihood objective similar to MMI (Graves et al., 2006). However, the LM probabilities are ignored,

$$\theta^* = \operatorname*{argmax}_{\theta} \sum_{n=1}^{N} \log P_\theta(\hat{\mathbf{w}}_n|\mathbf{X}_n) \tag{3.67}$$

$$= \sum_{n=1}^{N} \log \sum_{\mathbf{s}\in\mathcal{B}^{-1}(\hat{\mathbf{w}}_n)} \prod_t P_\theta(s_t|\mathbf{X}_n) \tag{3.71}$$

where s is the state sequence through the CTC-topology. Here, $\mathcal{B}$ is the set of state sequences that generate the word sequence $\hat{\mathbf{w}}_n$. The gradient of Equation (3.71) can be efficiently computed using the forward-backward algorithm and the result will resemble Equation (3.65). The benefit is that CTC models can be trained without any initial GMM system to obtain alignments.

3.10 Neural Transducers

A recently popularised ASR model is the neural transducer (Graves, 2012; Graves et al., 2013). It is an appealing model because it can be directly discriminatively trained. As explained before, every ASR system needs to have a component that can handle the fact that the input sequence is much longer than the output sequence. In HMM models, using a (possibly multi-state) topology with self-loops meant that more frames are consumed than output tokens. The CTC model (see Section 3.3.6) inserted blank symbols $\emptyset$ between output tokens and also had self-loops.

The neural transducer handles this differently, by treating consumption/emission of acoustic vectors separate from the emission of output symbols. Essentially, at each “time-step” the neural transducer decides whether or not to emit a symbol or to consume another acoustic vector. In a HMM or CTC model a symbol is emitted at the same time as an acoustic vector is consumed. Therefore, the output sequence of a neural transducer can be longer than the input acoustic sequence, which cannot be the case for a HMM or CTC model.

The transducer has an encoder $\varphi^{enc}(\cdot)$ that encodes all consumed acoustic vectors to generate the embedding,

$$\boldsymbol{e}_t = \varphi^{enc}\left(\mathbf{x}_{0:t-1}\right) \tag{3.72}$$

and a prediction network $\varphi^{pred(\cdot)}$ that is analogous to an Neural Network Language Model (NNLM) that generates an embedding $\boldsymbol{g}_m$ for the sequence of already emitted tokens $w_{0:m-1}$.

$$\boldsymbol{g}_m = \varphi^{pred}\left(w_{0:m-1}\right) \tag{3.73}$$

Based on $\boldsymbol{e}_t$ and $\boldsymbol{g}_m$ the decoder predicts $P(s'_{t,m}|\boldsymbol{e}_t, \boldsymbol{g}_m)$, which is a distribution over all symbols and consuming a frame. Because, $s_{t,m}$ does not have values for all t and m it effectively the length of the sequence of $s_{t,m}$ values is $T + M$. It also means that the neural transducer can handle sequence-to-sequence problems where $T < M$ *i.e.* it can output more symbols than there are acoustic frames, which is not possible for HMM or CTC approaches.

For training, the maximum likelihood objective function is used. To find the likelihood we need to sum over all $s_{t,m}$ "sequences" that generate the correct output sequence $\mathbf{w}$,

$$\begin{aligned}
\theta^* &= \underset{\theta}{\operatorname{argmax}} \sum_{n=1}^{N} \log P_\theta(\hat{\mathbf{w}}_n|\mathbf{X}_n) && (3.67)\\
&= \underset{\theta}{\operatorname{argmax}} \sum_{n=1}^{N} \log \left(\sum_{s \in \mathcal{B}^{-1}(\hat{\mathbf{w}}_n)} \prod_{(t,m)} P_\theta(s_{t,m}|\boldsymbol{e}_t, \boldsymbol{g}_m) \right) && (3.74)
\end{aligned}$$

where $\mathcal{B}^{-1}(\hat{\mathbf{w}}_n)$ is the set of all correct $s_{t,m}$ sequences and the multiplication is over all (t, m) pairs that exist in $\mathcal{B}^{-1}(\hat{\mathbf{w}}_n)$. As can be seen by comparing Equation (3.71) to Equation (3.74), the difference between the neural transducer and a CTC model is in the prediction network. The summation can be implemented using a modified version of the forward-backward algorithm. Popular choices of output tokens are graphemes or word pieces (see Table 3.1).

Neural transducer models are widely used in commercial applications due to their ability to naturally process audio in a streaming fashion (Li et al., 2021a; Rao et al., 2017; Saon et al., 2021).

3.11 Attention-Based Encoder-Decoder Models for ASR

The neural transducer can be trained directly using the maximum likelihood criterion. However, there are still independence assumptions being used. They have the advantage of allowing us to use the forward-backward algorithm, but can also limit modelling power. This section focuses on an ASR architecture that does not have any independence assumptions and can also be trained directly using the maximum likelihood criterion.

To achieve this a ANN, $\varphi(\cdot)$, is built that can directly model the distribution,

$$P_\theta\left(w_m|w_{1:m-1},\mathbf{X}\right)=\left[\varphi\left(w_{0:m-1},\mathbf{X}\right)\right]_{w_m} \tag{3.75}$$

where $[]_{\hat{w}_m}$ indicates taking the $\hat{w}_m$-th element of the output layer. This output layer uses a softmax output that has the same size as the number of possible tokens. $\varphi\left(w_{0:m-1},\mathbf{X}\right)$ thus transforms the history of tokens $w_{0:m-1}$ and the input acoustic vectors $\mathbf{X}$ into a distribution over the next token w_m. In this case the tokens w_m can be words, graphemes or word pieces. This allows the maximum likelihood criterion to be defined as,

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\sum_{n=1}^{N}\log P_\theta(\hat{\mathbf{w}}_n|\mathbf{X}_n) \tag{3.76}$$

$$= \underset{\theta}{\operatorname{argmax}}\sum_{n=1}^{N}\sum_{m=1}^{M+1}\log\left[\varphi\left(\hat{w}_{0:m-1},\mathbf{X}\right)\right]_{\hat{w}_m} \tag{3.77}$$

This means that the model can be trained using the CE loss of Equation (2.30) and the gradient of Equation (2.31). In general $\hat{\mathbf{w}}$ will include a start of sentence symbol as w_0 and a end of sentence symbol w_{M+1}.

This section will focus on $\varphi(\cdot)$ being represented using RNN-layers, though using Tranformers (Dong et al., 2018; Pham et al., 2019; Vaswani et al., 2017) and Conformers (Gulati et al., 2020; Guo et al., 2021; Li et al., 2021b) are also popular choices. In most cases, the ANN, $\varphi(\cdot)$, representing the ASR system will consist of three components joined together such that it is overall differentiable. It consists of an ANN encoder akin to the ANN in ANN-HMM hybrid models. This encoder usually consists of multiple RNN layers and maps the input sequence $\mathbf{x}_{1:T}$ to the sequence $\mathbf{h}^{enc}_{1:T'}$, where T' is often smaller than T. This will then consist of an alignment module that uses the attention mechanism of Section 2.1.6. It will generate a set of attention weights $\alpha_{m,1:T'}, m=1,\ldots,M+1$ for each output step m and thus also generate context vectors $\boldsymbol{c}_m=\sum_{i=1}^{T'}\alpha_{m,i}\mathbf{h}^{enc}_i$. This is followed by a decoder consisting of multiple RNN layers that will generate the final output distribution $\varphi\left(w_{0:m-1},\mathbf{X}\right)$.

One of the difficulties when dealing with sequence-to-sequence models can be the difference in input and output length. The average person speaks around 150 words per minute. Feature vectors are extracted every 10 ms and the number of graphemes per word is 6 on average. This results in the input sequence being around 7 times longer than the output sequence. Therefore, in order to reduce this ratio the RNN layers of the encoder will try to ensure that T' is smaller than T. This can be done by having RNN layers that operate at lower frequencies. For example, the first layer could take as input individual vectors $\mathbf{x}_t$. Then the second layer would take only every second output vector of the first layer as input. Then the third could take every second vector of the second layer as input. This would reduce the input length to the attention mechanism by a factor of 8 leading to roughly similar input and output lengths. Another method for reducing the input length is to concatenate multiple (*e.g.* three) input vectors.

The decoder operates in a very similar way to an RNN language model. The output is also a categorical distribution over output symbols obtained using the Softmax activation function. The difference is simply that it takes as input not only the previous output symbol but also the current context vector $\boldsymbol{c}_m$. Now, the input and the output sequence have the same length. This is the same as in an RNN language model, where the input is $w_{0:m}$ and the output is $w_{1:m+1}$. For the decoder, the input sequence is $[(w_0, \boldsymbol{c}_1), (w_0, \boldsymbol{c}_1) \dots (w_m, \boldsymbol{c}_{m+1})]$.

However, the decoder when used for recognition needs to actually generate a sequence $\mathbf{w}^\star$ rather than simply assign probabilities. As mentioned before the first output w_0 is fixed. Based on w_0 and $\boldsymbol{c}_1$ the decoding process begins. At each output time step m the decoder outputs a categorical distribution over output tokens, from which the next most probable output is taken as w_m:

$$\begin{aligned} P_\theta\left(w_m|w_{1:m-1}, \mathbf{X}\right) &= \left[\varphi\left(w_{0:m-1}, \mathbf{X}\right)\right]_{w_m} && (3.78)\\ &= \left[\varphi^{dec}\left(\boldsymbol{c}_{1:m-1}, w_{0:m-1}\right)\right]\Big]_{w_m} && (3.79)\\ w_m^\star &= \arg\max_{w'_m} P_\theta\left(w'_m|w_{0:m-1}, \mathbf{x}_{1:T}\right) && (3.80) \end{aligned}$$

When the RNN decoder generates the end of sentence token, the decoding process is terminated.

Lastly, consider the attention mechanism, the methods to obtain $\alpha_{m,1:T'}, m = 1, \dots, M$ where T' is the output length of the RNN encoder. The attention mechanism essentially performs an alignment process between the output of the encoded sequence $\mathbf{h}_{1:T'}$ and the output sequence. The weights $\alpha_{m,i}$ should give weight to parts of the input that are relevant to the token that is currently being generated. The information needed to compute the $\alpha_{m,i}$ values thus consists of the encoded sequence $\mathbf{h}^{enc}_{1:T'}$ and the output history represented by the RNN decoder state $\mathbf{h}^{dec}_{m-1}$[4]. There are various methods to generate the weights $\alpha_{m,1:T'}$ one of which was described in Section 2.1.6. Two further methods, that are popular in ASR are given here. First,

$$e_{m,i} = \varphi^d\left(\mathbf{h}^{enc}_i\right)^T \varphi^e\left(\mathbf{h}^{dec}_{m-1}\right) \tag{3.81}$$

where φ^d and φ^e are MLPs as used by Chan et al. (2016).

[4]The previous focus *i.e.* the previous $\alpha_{m-1,i}$ can also be used as used by Watanabe et al. (2017)

Second,

$$e_{m,i} = g^T \left(\boldsymbol{W} \mathbf{h}_i^{enc} + \boldsymbol{V} \mathbf{h}_{m-1}^{enc} + b \right) \tag{3.82}$$

where $\boldsymbol{W}$ and $\boldsymbol{V}$ are learned weight matrices and g and b are learned vectors, which was used by Watanabe et al. (2017).

In both cases,

$$\alpha_{m,i} = \frac{\exp(e_{m,i})}{\sum_{j=1}^{T'} \exp(e_{m,j})}, i = 1, \ldots, T' \tag{3.83}$$

to form the distribution over output units.

3.11.1 Beam-Search Decoding for AED models

The decoding process described above (Equation (3.80)), which only considers a single possible output history, is called *greedy decoding*. A better form of decoding that considers multiple output histories is called *beam-search*. In beam-search with n histories (called beams), the outputs corresponding to the top n probabilities of $P_\theta(w_1|w_0, \mathbf{x}_{1:T})$ are taken as the first n beams. For each succeeding w_m, the n outputs corresponding to the top n probabilities of $P_\theta(w_m|w_{0:m-1}, \mathbf{x}_{1:T})$ are taken for each of the n beams. This is followed by reducing the number of beams from now n^2 down to n, by taking the n beams with the highest probability $P_\theta(w_{1:m}|\mathbf{x}_{1:T}, w_0)$. This means that the decoder hidden state can be different for each beam.

This decoder can output potentially infinitely long sequences. Hence, a maximum output sequence length is usually set. For ASR we can use the number of acoustic vectors T, since in almost all settings the duration of an output token is less than the duration of an acoustic frame.

Furthermore, similar to HMM based models an insertion penalty ι is used. Furthermore, to favour beams that use the entire input sequence, a coverage coefficient ω is introduced, that favours beams that focus on a larger number embedding vectors $\mathbf{h}_{1:T'}^{enc}$ (Chorowski and Jaitly, 2017). An embedding vector is "focused on" if the cumulative attention it receives is above a threshold τ. Combining this yields

$$\mathbf{w}^\star = \arg\max_{\mathbf{w}} \log P_\theta(\mathbf{w}|\mathbf{X}) + \iota \|\mathbf{w}\| + \omega \sum_{i=1}^{T'} \left[(\sum_{m=1}^{M+1} \alpha_{m,t}) > \tau \right] \tag{3.84}$$

3.11.2 Scheduled Sampling

The ML-training of these models is based on using CE training for the distribution,

$$P_\theta(w_m|w_{1:m-1}, \mathbf{X}) = [\varphi(w_{0:m-1}, \mathbf{X})]_{w_m} \tag{3.75}$$

here the history sequence $w_{0:m-1}$ is taken as the ground-truth sequence from the data set. During decoding, however, $w_{0:m-1}$ will often have errors. To overcome this exposure bias, scheduled sampling (Bengio et al., 2015) can be used. In scheduled sampling, with a certain probability, not the ground truth token is taken for the history, but the token with the highest probability of the output distribution. This probability is often changed during training. Usually, it is zero at the start of training and then increased throughout training.

3.12 Summary

This chapter introduces the main methods for building ASR systems. These include the front-end processing methods, such as PLP and FBANK features that will be used in this thesis as the input to GMM and ANN models, respectively. Further, HMM-GMM models were described in detail, which will be used within the active learning Chapters 7 and 8 of this thesis. The CTC approach was introduced, which is used in Chapter 9. Finally, unsupervised HMM-GMM models are used in Chapter 9 to provide an unsupervised tokenisation algorithm. The three-state HMM structure that is explained in this chapter provides an inspiration to the minimum-duration HMM proposed in Chapter 9.

Chapter 4

Semi-Supervised Learning

Semi-supervised learning concerns learning from both labelled and unlabelled data in order to improve a model that normally would learn from labelled data only (Chapelle et al., 2006). While labelled data is difficult to obtain in most domains, unlabelled data is often available in large quantities and relatively easy to collect. However, traditional supervised learning methods cannot use unlabelled data for training models. Semi-supervised learning is attractive because it can potentially utilise both labelled and unlabelled data to achieve better performance than supervised learning. In turn, this means that semi-supervised learning can result in a similarly (or more) accurate model but trained on far less supervised data [1]. This reduces the labelling cost, which leads to reduced costs overall.

The training data consists of both L labelled instances $\mathcal{D}_l = \{(\mathbf{x}_i; \hat{\mathbf{y}}_i)\}_{i=1}^{L}$ and U unlabelled instances $\mathcal{D}_{ul} = \{\mathbf{x}_i\}_{i=L+1}^{L+U}$. In classification, the labels are discrete. In sequence-to-sequence modelling such as speech recognition, $\mathbf{x}_i$ is an input sequence of real-valued vectors and $\mathbf{y}_i$ is an output sequence of discrete values. It is typically assumed that there is much more unlabelled data than labelled data, i.e., $U >> L$. The goal of semi-supervised learning is to train from both the labelled and unlabelled data, such that it is better than a supervised learner trained on the labelled data alone.

At first, it might seem paradoxical to learn anything about a mapping $f : \mathcal{X} \to \mathcal{Y}$ from unlabelled data. The function f describes the mapping between the input $\mathbf{x}$ and the output label y, but the unlabelled data set does not contain any examples of this mapping. However, the key to understanding this lies in the assumptions made about the connection between the distribution of unlabelled data $p(\mathbf{x})$ and the target label.

Figure 4.1 (taken from Oliver et al. (2018)) shows an example of semi-supervised learning. The performance of various semi-supervised learning approaches for Multilayer Perceptrons (MLPs) was evaluated on the "two moons" data set. This data set contains two classes,

[1] For example, see Appendix C in (Mohamed et al., 2022)

each following a half-moon distribution. The two half-moons are intertwined, making it a challenging task that requires a non-linear decision boundary. In this semi-supervised learning scenario, three labelled data points are provided for Class 1 and three labelled data points for Class 2, along with a large number of unlabelled data points. Using supervised learning alone, the resulting decision boundary is shown by the dashed line. With only these six data points, the supervised decision boundary cannot capture the non-linear nature of the data. By assuming that the data forms coherent groups ($i.e.$ the data naturally groups together into contiguous high-density regions of individual classes), the non-linear nature of the data can be captured. The blue and yellow lines, produced by semi-supervised learning techniques that apply this assumption, draw the decision boundary in regions with fewer data points and are therefore able to capture the non-linear nature of the data. The assumption of coherent groups works because the data does not have a high degree of class overlap. The amount of class overlap, in part, determines how effective semi-supervised learning will be in comparison to supervised learning (Castelli and Cover, 1996; O'Neill, 1978).

This chapter describes several semi-supervised learning techniques. Each technique is based on different assumptions about the relationship between the input data distribution $p(\mathbf{x})$ and the class-conditional distribution $P(y|\mathbf{x})$. These techniques can be applied to a variety of problems and have been shown to produce better classifiers than supervised learning alone under specific training conditions on specific data sets. It is important to note that not every semi-supervised learning method will necessarily improve performance over supervised learning for every task. In some cases, the use of unlabelled data can even result in worse performance if the incorrect semi-supervised learning algorithm is used (Cozman et al., 2003). Section 4.8 describes a class of unsupervised learning methods known as "self-supervised representation learning", which is used to learn a better representation of the data using only the unsupervised data. This separates semi-supervised learning into two steps, where first, the representation is learned, and then the supervised model is trained using the improved representation as input. In some cases, the encoder that provides the representation is also fine-tuned during supervised training.

4.1 Semi-supervised learning in ASR

In the semi-supervised setting, the Automatic Speech Recognition (ASR) system and its learning algorithm has access to both a data set containing labelled utterances and also to a (usually much larger) corpus of utterances for which only the audio is available. In many cases, this audio is not even split into utterances and might, for example, come in the form of the

[2]Figure taken from (Oliver et al., 2018)

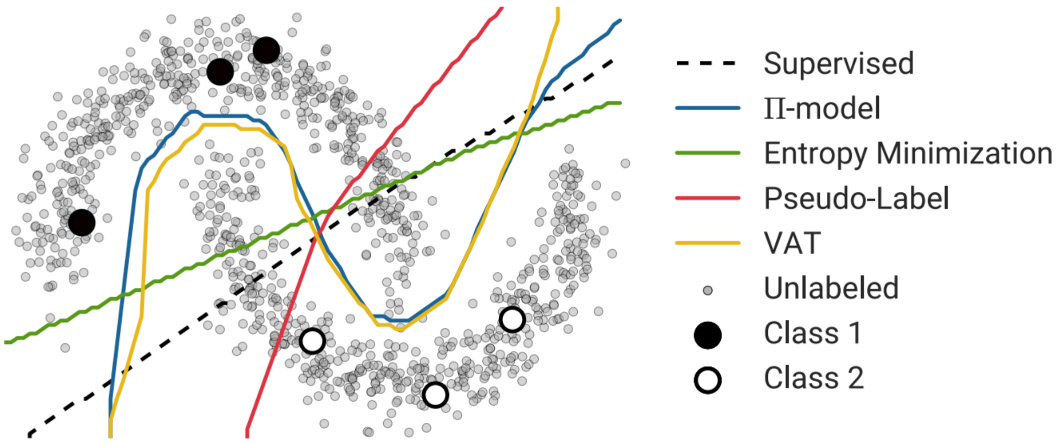


Figure 4.1 Comparison of the decision boundaries drawn by an MLP trained with different Semi-Supervised learning approaches[2].

audio track of entire TV shows. Semi-supervised learning for speech recognition aims to learn from both transcribed and untranscribed data.

The value of semi-supervised learning in ASR is rather obvious. A near-infinite amount of unsupervised data (pure audio) is available from numerous sources. For languages for which collecting thousands of hours of transcribed audio is not commercially viable, untranscribed data can still be collected relatively cheaply by comparison. The use of semi-supervised learning also offers privacy advantages, as privacy regulations relating to the use of transcribed data are often stricter than those relating to untranscribed data.

Even though many companies are trying to build general-purpose ASR systems that can be used in many settings, it is still normally the case that building an ASR system for a specific purpose yields superior performance. When building an ASR system for a specific purpose, task-specific data is required. However, most unsupervised data that exists is probably not relevant, it is not *in-domain* data but rather *out-of-domain* data. Therefore, when trying to acquire suitable untranscribed audio, it is not good enough to simply take a random selection of YouTube videos or TV-broadcasts, since the untranscribed data has to be curated for the task at hand. Even if this problem is solved, then ideally the available out-of-domain *transcribed* data should also be used. This causes a shift in the marginal distribution $p(\mathbf{x})$ between the different types of data: transcribed in-domain data, transcribed out-of-domain data and untranscribed out-of-domain data. Most theory on machine learning is concerned with situations where there is no distributional shift and the literature on semi-supervised learning under domain shifts is comparatively sparse.

In ASR an example of a scenario in which there is a mismatch between the supervised and the unsupervised scenario is training on labelled read-speech and unlabelled spontaneous conversational speech. This is useful because read-speech has much lower labelling costs than spontaneous conversational speech. Such a scenario was investigated by Wang et al. (2007) in which the supervised data was mostly Mandarin broadcast news (predominantly read speech), and the unsupervised data was a mix of Mandarin broadcast news and Mandarin broadcast conversational speech. The improvements of the ASR model from the semi-supervised learning, when tested on a Mandarin broadcast conversational test set were very small (2.4% relative Character Error Rate reduction). In Chan et al. (2021) an ASR system used a large encoder trained on read-speech (Librispeech (Panayotov et al., 2015)). The ASR system was then trained using supervised learning on a multi-domain data set and tested on multiple data sets. Only on the test sets containing read speech (*i.e.* similar to the unsupervised data; *e.g.* Librispeech (Panayotov et al., 2015) or CommonVoice (Ardila et al., 2020)) was the performance better than without the unsupervised encoder. On test sets containing spontaneous conversational speech (*e.g.* the SwitchBoard (SWB) or CallHome (CH) portions of Hub5'00) the performance was worse. This shows that semi-supervised learning with a mismatched scenario between the supervised and the unsupervised case is an unsolved problem that needs further investigation.

Semi-supervised learning for ASR can also take the form of having additional language modelling data without corresponding audio instead of having audio without the corresponding word sequence. Using speech synthesis systems the additional language data can be transformed into ASR training data (Wang et al., 2020b). This was not studied in this thesis.

4.2 Semi-Supervised and Unsupervised Learning

Unsupervised learning algorithms are those algorithms that only use the unlabelled data set $\mathcal{D}_{ul} = \{\mathbf{x}_i\}_{i=L+1}^{L+U}$. Standard approaches to clustering are examples of unsupervised learning algorithms. Clustering is the task of grouping data samples into multiple clusters such that a sample of a particular cluster is more similar to samples of that cluster than to those of other clusters. Another example would be learning the underlying distribution of the data using methods such as factor analysis or a Gaussian Mixture Model (GMM). In modern deep learning, unsupervised representation learning (see Section 4.8) is very popular. Such methods attempt to learn a better representation of the data points after which a supervised model is trained on those representations.

Semi-supervised learning falls somewhere between unsupervised and supervised learning. In fact, many semi-supervised learning strategies are based on extending either unsupervised or supervised learning to include additional information typical of the other learning paradigm. Some semi-supervised learning algorithms even combine an unsupervised learning loss with a supervised learning loss by defining a cost-function $\mathcal{C}_{\mathrm{US}}(\mathcal{D}_{ul}, \theta)$ for the unlabelled data and combining it with the supervised learning cost function $\mathcal{C}_{\mathrm{S}}(\mathcal{D}_{l}, \theta)$, such as the Cross Entropy (CE) objective:

$$\mathcal{C}_{\mathrm{SSL}}(\theta) = \mathcal{C}_{\mathrm{S}}(\mathcal{D}_{l}, \theta) + \alpha \cdot \mathcal{C}_{\mathrm{US}}(\mathcal{D}_{ul}, \theta) \tag{4.1}$$

where α is an interpolation coefficient. However, this does not mean that training purely on $\mathcal{C}_{\mathrm{US}}(\mathcal{D}_{ul}, \theta)$ would constitute unsupervised learning. The main difference between the part of semi-supervised learning algorithms that trains on the unlabelled data and pure unsupervised learning algorithms is that often training on the unlabelled data alone is somewhat meaningless and would not yield a usable parameter estimate. The proposed methods investigated in Chapter 6 are of the type that is presented in Equation (4.1).

Many other semi-supervised learning algorithms rely on a model that is pre-trained using supervised learning, after which there is a further training stage on the unsupervised data. Here, training on the unsupervised portion would also not be considered unsupervised learning as without the pre-trained model, the learning from the unsupervised data would not lead to better parameter estimates.

4.3 Transductive vs Inductive Semi-Supervised Learning

Semi-supervised learning encompasses two distinct modes: inductive and transductive semi-supervised learning. In supervised classification with a fully labelled training set, the objective is to achieve high accuracy on future test data. In semi-supervised classification, however, the training data contains some unlabelled data, and two different problems can be posed. Even though the training data might be fully labelled, the test data is not. This is the case in transductive semi-supervised learning, where the unlabelled test samples are included in the training set, and the task is to predict their labels. This can be seen to be similar to unsupervised *test-time adaptation* in speech recognition (Gales, 1998; Leggetter and Woodland, 1995), but in this case, all the test data is available.

The most prevalent scenario of semi-supervised learning is *inductive* semi-supervised learning. In this scenario, the training set consists of both labelled and unlabelled data. A classifier is trained and then tested on unseen data as if the classifier were trained on labelled data only. It is possible to combine inductive semi-supervised learning with transductive semi-supervised learning. In the sections hereafter, this chapter will focus solely on inductive semi-supervised learning. Examples of transductive learning include graph-based methods such as Graph Mincut (Blum and Chawla, 2001) or Harmonic Functions (Zhu et al., 2003).

4.4 Self-Training

The simplest and easiest to use semi-supervised learning technique is self-training. In self-training the model's own predictions are used to teach the model. For this reason, it is also called self-teaching. Self-training can be either inductive or transductive, depending on how it is applied.

The model is first trained on the labelled data. The model is then used to predict labels for the unlabelled data (class for a classifier; transcriptions for an ASR model). The unlabelled data with the new labels are then used to augment the labelled data. In many cases, only a subset of the unlabelled data, together with its labels, is selected (see Section 4.4.1). Typically this subset is selected in such a way that the subset consists of the unlabelled samples with the most confident predictions. The model is then re-trained on the now larger set of labelled data. This procedure can then be repeated, and the labels are predicted again for the unlabelled data. The first successful applications of self-training were speech recognition (Kemp and Waibel, 1998; Zavaliagkos and Colthurst, 1998) and word-sense disambiguation (Yarowsky, 1995). Self-training is also sometimes called "pseudo-labelling" (Lee, 2013) as in Figure 4.1. In some use cases it makes sense to use one model (typically a larger model *e.g.* Xiao et al. (2022)) to predict labels (or transcriptions) for the unlabelled data which are then used to train another model, in which case the term pseudo-labelling is more appropriate. For self-training of an ASR system, the strength of the language model used during decoding has a significant impact, as shown by Wallington et al. (2021).

4.4.1 Utterance-Transcription Pair Selection

Often the assumption is made that training on only a reasonably well decoded portion of the data and not training on badly decoded data leads to better performance. Therefore the new automatically transcribed audio-data is filtered with the aim of retaining only the accurately decoded data. This is often done based on a confidence-measure, by simply excluding those portions of the audio on which the system is not confident.

Zavaliagkos and Colthurst (1998) proposed this method but their confidence measure would have excluded a too large proportion of the data. Hence, they enlarged their filtered data set of audio and automatic transcriptions by correcting some of the automatic transcriptions using the ground truth transcriptions of the unlabelled audio. Kemp and Waibel (1998, 1999) were the first to build a full system trained in the semi-supervised setting without using the true transcription of the unlabelled audio. The ability to do so could be ascribed to the use of a significantly improved method to compute the confidence-measure. Wessel and Ney (2001, 2005) corroborated these results with experiments on a larger data set and also used the final model to re-decode the unlabelled audio in order to then re-train the model on the new automatic transcriptions, which were accurate than the transcriptions given by the seed model. Lamel et al. (2002a) proposed this iterative training procedure at about the same time as Wessel and Ney (2001).

One challenging type of error that recognisers make are deletions. These errors are not accounted for by the standard confidence estimation schemes (Seigel and Woodland, 2014) and are hard to rectify in the upstream and downstream processing. Confidence estimation can be combined with deletion prediction for improved filtering (Ragni et al., 2018).

Self-training can be improved by using multiple iterations and by using data augmentation (Park et al., 2020).

4.4.2 Minimum Entropy Approaches

The additional cost function C_{US} for self-training can be defined as the log-probability on the predictions made by the same model with the previous parameter setting:

$$C_{\text{US}}(\theta) = \sum_{i=L+1}^{L+U} \log P_\theta\left(y_i^\star \mid \mathbf{x}_i\right) \tag{4.2}$$

$$\text{where} \quad y_i^\star = \arg\max_y \; P_{\theta'}\left(y \mid \mathbf{x}_i\right) \tag{4.3}$$

where θ' is the previous setting of θ. For the first iteration of self-training, θ' is the parameter setting after training purely on the supervised data. For later iterations, θ' is the parameter setting from the previous training iteration. If instead of taking the subset of unlabelled samples with the highest confidence, each sample is weighted by its respective confidence and the iterative connection between θ' and θ is removed, the following cost function results:

$$C_{\text{US}}(\theta) = \sum_{i=L+1}^{U+L} \sum_{k=1}^{K} P_\theta\left(y_k \mid \mathbf{x}_i\right) \log P_\theta\left(y_k \mid \mathbf{x}_i\right) \tag{4.4}$$

$$= \sum_{i=L+1}^{U+L} \mathbb{H}_{P_\theta(y|\mathbf{x}_i)}\left[y|\mathbf{x}_i\right] \tag{4.5}$$

where $\mathbb{H}(\cdot)$ is the entropy. As shown in Equation (4.5), the resulting cost function is the *minimum entropy* criterion, which was proposed for semi-supervised learning by Grandvalet and Bengio (2004). Entropy minimisation encourages the classifier to be confident on all samples, whilst self-training with a filtering approach encourages high confidence only on samples where the classifier is already confident. The disadvantage of entropy minimisation is that when minimising C_{US} using a gradient-based method, the gradients w.r.t. θ have to exist. For standard self-training approaches, the classifier can be any arbitrary classifier.

For ASR systems, the *minimum entropy* criterion can be derived. When training using the Maximum Mutual Information (MMI) criterion (see Section 3.9.3) in combination with self-training, the following cost function is minimised for the unlabelled data set for a sequence model:

$$\begin{aligned} C_{\text{MMI}}(\theta) &= \sum_{i=L+1}^{U+L} \log P_{\theta}(\mathbf{w}_i^{\star}|\mathbf{X}_i) \\ \text{where} \quad \mathbf{w}_i^{\star} &= \arg\max_{\mathbf{w}} P_{\theta'}(\mathbf{w}|\mathbf{X}_i) \end{aligned} \tag{4.6}$$

where θ' is the previous parameter setting of the model. Similar to the classification case, a confidence-based weighting of the utterances leads to the following objective:

$$C_{\text{Weighted MMI}}(\theta) = \sum_{i=L+1}^{L+U} P_{\theta}(\mathbf{w}_i^{\star}|\mathbf{X}_i) \log P_{\theta}(\mathbf{w}_i^{\star}|\mathbf{X}_i) \tag{4.7}$$

If all possible output hypotheses are considered instead of only the one most probable, this leads to the Negative Conditional Entropy criterion (Huang and Hasegawa-Johnson, 2010; Manohar et al., 2018, 2015):

$$C_{\text{Cond. Entropy}}(\theta) = \sum_{i=1}^{U} \sum_{\mathbf{w}\in\mathcal{W}_i} P_{\theta}(\mathbf{w}|\mathbf{X}_i, \theta) \log P_{\theta}(\mathbf{w}|\mathbf{X}_i) \tag{4.8}$$

where $\mathcal{W}_i$ is the set of all possible output hypotheses for the input $\mathbf{X}_i$. This conditional entropy framework is the ASR equivalent of the entropy minimisation framework by Grandvalet and Bengio (2004). In general, self-training and entropy minimisation are very similar as both encourage high-confidence (low-entropy) predictions on unlabelled data. When using confidence-based filtering, the difference becomes that high-confidence predictions are only encouraged on those points on which the model already has high confidence.

4.5 Lightly-Supervised Training

Related to semi-supervised training is an area referred to as *lightly-supervised* or *weakly-supervised* training. Lightly-supervised is usually described as the setting in which training data is provided for which transcriptions are available which are known to be noisy. For example, the main US television channels also broadcast manually derived closed-captions. These closed-captions, while accurately reflecting the meaning of the utterance, are often not literal transcriptions of the data. Hesitations and repetitions are not marked, and there may be word insertions, deletions and even changes in the word order. There may also exist other sources of information with different levels of completeness, such as approximate transcriptions, summaries or keywords, which can be used to provide some supervision.

A different filtering approach can be used in the lightly supervised setting than in standard self-training, by using only the automatic transcriptions that match the, possibly erroneous, closed-captions of television or radio broadcast data. This was first done by Lamel et al. (2000, 2001, 2002b). Lamel et al. (2001) demonstrated that the filtering is not necessary as long as the close-captions are included in the language model used to decode the unlabelled audio. It was proposed by Chan and Woodland (2004) to bias the language model used for decoding towards the closed-captions of the unlabelled audio and also showed that self-training works using the MMI and Minimum Phone Error (MPE) training objectives.

4.6 Cluster-then-Label Approaches

Probabilistic generative models apply the Expectation Maximisation (EM) algorithm, where for the unsupervised data, first, (soft) class assignments are derived $P_\theta(y = k|, \mathbf{x}_i)$ and then class-specific generative models $p_\theta(\mathbf{x}_i, y = k)$ are optimised. Similarly, another straightforward way to perform semi-supervised learning is to first identify clusters using unlabelled data and some unsupervised clustering algorithm. Then based on the labelled data, the individual clusters are assigned to a specific class. This high-level idea is often referred to as a *Cluster-then-Label Approach*. Under certain theoretical conditions, theoretical analysis also justifies the Cluster-then-label procedure (Demiriz et al., 1999). This approach has been successfully applied to deep learning for the MNIST data set (Haeusser et al., 2018; Ji et al., 2018; Kosiorek et al., 2019). For deep learning, Artificial Neural Networks (ANNs) are trained and the values from one of the hidden layers is used to apply a clustering algorithm such as K-means. Kosiorek et al. (2019) performs purely unsupervised clustering (K-means on hidden vectors), but the cluster assignment is based on all the data by finding and assigning a cluster the label of the hidden vector in the training set that is closest to the centroid of the cluster. An accuracy of 98.9% was reported on the official MNIST train/test split. At the same time, however, an accuracy of 19.39% was reported for the CIFAR-10 data set, showing that these algorithms do not yet perform well on complex data sets (CIFAR-10 is more complex than MNIST). Even when the extracted hidden vectors are classified using a linear classifier trained on all the supervised data, the accuracy is only 33.48% (98.9% on MNIST). It is likely that these approaches could benefit from a semi-supervised approach to extracting the hidden representation. Clustering is hard if it is not known along which dimension to cluster. In an ASR data set, the data could be clustered based on semantic content, the speaker's emotion, the speaker itself, background noise conditions or the accent of the speaker. Without explicit supervision obtaining a disentangled representation such that the clustering is useful in the desired downstream classification class, explicit supervision or strong inductive biases are needed (Locatello et al., 2019). Cluster-then-Label approaches will be revisited in Section 4.8 where they inspired another range of semi-supervised learning algorithms.

4.7 Consistency Regularisation

Consistency regularisation is a regulariser (see Section 2.3) that belongs to the group of regularisers that adds an auxiliary cost function to the supervised learning loss. To calculate the consistency regularisation loss, multiple versions of the same unlabelled data point are passed through the same classifier. The auxiliary cost function is then the disagreement between these predictions. The difference in prediction can be defined by any distance measure between distributions. Usually, the Kullback-Leibler divergence (KL-divergence) divergence is used, but also the squared distance between outputs or activations is used.

Examples of consistency regularisation methods for deep learning are the Π model, virtual adversarial training (VAT) and Ladder Networks (Miyato et al., 2019; Rasmus et al., 2015; Valpola, 2015). The Π model builds multiple inputs based on stochastic regularisation techniques *i.e.* different drop-out masks are used, or data-augmentation techniques (such as different crops or rotations of images) are used. VAT tries to build a second input that increases the Kullback-Leibler divergence (KL-divergence) between the two inputs the most. VAT inspires the work in Chapter 6, is briefly explained below and in more detail in Chapter 6.

Sometimes these techniques are also called *multi-view training* (Sindhwani et al., 2005), although this usually refers to data sets in which the same unlabelled data point is actually observed from two different views. This could be an image from different angles or the text from a website together with the text from websites that link to the same website.

4.7.1 Virtual Adversarial Training

A popular belief, based on widely observed facts is that the outputs of most naturally occurring systems are smooth with respect to spatial and temporal inputs. In VAT (Miyato et al., 2019), an additional loss is introduced, which tries to smooth the output distribution isotropically around every data point that is part of the supervised and the unsupervised training data.

In VAT, the input is perturbed in such a way that the output distribution changes the most. The distance measure to determine the change is the Kullback-Leibler divergence (KL-divergence) divergence. The additional cost $\mathcal{C}_{\text{US}} = \mathcal{R}_{\text{VAT}}(\mathcal{D}_l, \mathcal{D}_{ul}, \theta)$ is then the KL-divergence between the output distributions for the perturbed and the unperturbed inputs.

$$\mathcal{R}_{\text{VAT}}(\mathcal{D}_l, \mathcal{D}_{ul}, \theta) = \frac{1}{L+U} \sum_{\mathbf{x} \in \mathcal{D}_l, \mathcal{D}_{ul}} \text{LDS}(\mathbf{x}, \theta) \tag{4.9}$$

$$\text{LDS}(\mathbf{x}, \theta) = D[P_{\theta'}(y \mid \mathbf{x}), P_{\theta}(y \mid \mathbf{x} + \mathbf{r}_{\text{VAT}})]\Big|_{\theta' = \theta} \tag{4.10}$$

where $D[\cdot, \cdot]$ is the KL-divergence. LDS$(\cdot, \cdot)$ stands for local distributional smoothness. θ' is a fixed copy of the current parameters of the ANN *i.e.* the backpropagation algorithm is only applied to the second term in the KL-divergence divergence of Equation (4.10). $\mathbf{r}_{\text{VAT}}$ is the virtual adversarial perturbation.

$$\mathbf{r}_{\text{VAT}} = \arg\max_{\mathbf{r}; \|\mathbf{r}\| \leq \epsilon} D[P_{\theta}(y \mid \mathbf{x}), P_{\theta}(y \mid \mathbf{x} + \mathbf{r})] \tag{4.11}$$

i.e. the perturbation that increases the KL-divergence the most. Details on how $\mathbf{r}_{\text{VAT}}$ is calculated are given in Chapter 6.

4.8 Unsupervised Representation learning

As alluded to in Chapter 2 in Figure 2.1, drawing decision boundaries for a specific task is a much easier task once a good representation has been learned. This section is concerned with how to generate a good representation without using any labelled data.

Most mainstream approaches fall into one of two classes: generative or discriminative. Generative approaches learn to generate or otherwise model the input space. The fast learning algorithm by Hinton et al. (2006) for the Restricted Boltzmann Machine (RBM) is an early example of generative unsupervised representation learning. This algorithm, however, was only used to initialise the weights of Deep Neural Networks (DNNs). Later, the encoder of a Variational Auto-Encoders (VAEs) (Kingma and Welling, 2014) was used as a representation for a Support Vector Machine (SVM) by Kingma et al. (2014). In ASR, encoder-decoder models have been trained to encode a noisy waveform and reconstruct clean input features from the encoding (Pascual et al., 2019; Ravanelli et al., 2020). However, direct modelling of the input space is expensive and is most likely not necessary for representation learning.

Discriminative approaches learn representations using objective functions similar to those used for supervised learning, but train networks to perform pretext tasks where both the inputs and labels are derived from an unlabelled data set. For example, in Doersch et al. (2015) a model is trained to predict the relative (8-connected) position of two patches within an image.

Both patches are passed through the same encoder and the relative position (a number between 1 and 8) is predicted based on the two embeddings. Noroozi and Favaro (2016) built upon this and fed 9 patches through the same encoder to obtain embeddings. These embeddings were permuted and a network was then trained to predict the original positions (*e.g.* the third patch should be in position 1 (top left corner)). Given that the total number of possible permutations is $9! \approx 4 \cdot 10^5$, only a subset of 1000 permutations was chosen. Gidaris et al. (2018) used the relative rotation between two images as the target. Grill et al. (2020) proposed a teacher-student learning approach where the teacher and student have the same structure and the student is trained to predict the same vector as the teacher for augmented versions of the input. At the same time, the parameters of the teacher are an exponentially smoothed version of the student's parameters. In the DeepCluster method (Caron et al., 2018), which is based on a cluster-then-label approach (see Section 4.6), the embedding of an ANN is first clustered and then the cluster assignments are used as targets for supervised training. This approach is run iteratively.

Another line of discriminative representation learning has evolved that uses contrastive losses. For a contrastive loss, for each input data point, there is a positive example and negative examples. The contrastive loss ensures that i) the embedding of the data point is similar to the embedding of the positive example and ii) the embeddings of the data point is dissimilar to the embeddings of the negative examples. Figure 4.2 gives an illustration of a contrastive learning algorithm called SimCLR (Chen et al., 2020). An input vector $\mathbf{x}_i$ is transformed using data augmentation techniques tr and tr' into $\tilde{\mathbf{x}}_{2i}$ and $\tilde{\mathbf{x}}_{2i+1}$, respectively. An ANN $\phi^1(\cdot)$ transforms the augmented inputs into $\mathbf{h}_{2i}$ and $\mathbf{h}_{2i+1}$. It is this ANN, $\phi^1(\cdot)$, that is used in later stages (*e.g.* supervised fine-tuning). Another ANN $\phi^2(\cdot)$ then transforms the embeddings into $\mathbf{z}_{2i}$ and $\mathbf{z}_{2i+1}$. Given that the augmentation techniques are chosen such that they do not change any information about the input vector that should be kept, the loss should encourage these two embeddings to be the same. The loss is given by:

$$\mathrm{sim}'(\mathbf{x}, \mathbf{y}) = \exp(\mathrm{sim}(\mathbf{x}, \mathbf{y}))/\tau \tag{4.12}$$

$$l_i = -\log \frac{\mathrm{sim}'(\mathbf{z}_{2i}, \mathbf{z}_{2i+1})}{\sum_{k=0}^{2B-1} \mathbb{1}_{[2i \neq k]} \mathrm{sim}'(\mathbf{z}_{2i}, \mathbf{z}_k)} - \log \frac{\mathrm{sim}'(\mathbf{z}_{2i+1}, \mathbf{z}_{2i})}{\sum_{k=0}^{2B-1} \mathbb{1}_{[2i+1 \neq k]} \mathrm{sim}'(\mathbf{z}_{2i+1}, \mathbf{z}_k)} \tag{4.13}$$

where B is the size of the batch that is being considered, τ is a temperature parameter and $\mathbb{1}_{[2i \neq k]} \in \{0, 1\}$ is an indicator function evaluating to 1 iff $2i \neq k$. $\mathrm{sim}(\cdot, \cdot)$ is a similarity measure, for example, the cosine similarity. The numerator of the loss pushes the embeddings of two distorted samples closer together, and the denominator pushes the embeddings of two different samples further apart.

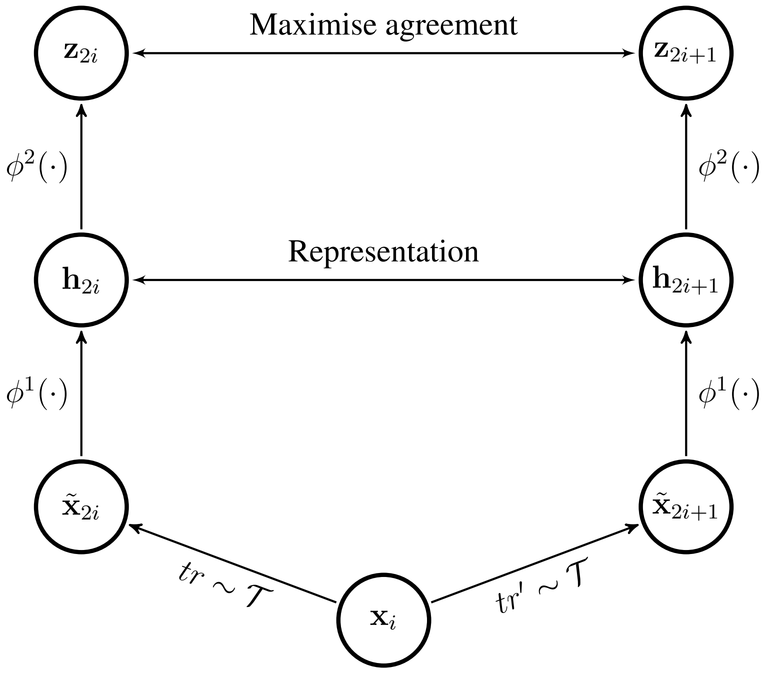


Figure 4.2 Illustration of the SimCLR framework (Chen et al., 2020). $\mathbf{x}_i$ is an unlabelled data point and tr and tr' are data transformations randomly chosen from the set of data augmentation schemes $\mathcal{T}$.

Similar approaches have been proposed by He et al. (2020) and Wu et al. (2018). In Jiang et al. (2021) Khorram et al. (2022) and Baevski et al. (2022) such approaches were tried for ASR. The different data augmentation methods that were tried in these three papers were various audio-based methods such as pitch shift, reverberations and additive noise for both tr and tr' in Jiang et al. (2021). Khorram et al. (2022) used speed perturbation for only tr and took care to align the resulting embeddings. Baevski et al. (2022) uses time-masking for tr only.

4.8.1 BERT

Standard Recurrent Neural Network (RNN)-Language Models (LMs) are trained using the cross-entropy loss for the conditional probability $P(w_i|w_{0:i-1})$. It has been found that using the embeddings of such LM *i.e.* the vectors of the penultimate layer are good embeddings for a variety of Natural Language Processing (NLP) tasks (Howard and Ruder, 2018; Peters et al., 2018; Radford et al., 2018) such as Question Answering, Natural Language Inference or paraphrasing. Even though for LMs future words and hence bi-directional models need to be used with care (because LMs model the probability $P(w_{1:N}) = \prod_{i=1}^{N} P(w_i|w_{0:i-1})$), bi-directional models can be used to obtain embeddings for NLP tasks.

Devlin et al. (2019) introduced Bidirectional Encoder Representations from Transformers (BERT). BERT is not trained to approximate the distribution $P(w_i|w_{0:i-1})$, but instead it uses a masked language model objective *i.e.* certain words are masked in the input sequence and the model is trained to predict those using the cross-entropy loss at the masked positions. In Devlin et al. (2019) 15% of the input tokens (30k word-pieces) were masked. Some additional design choices were made for better performance. In order to avoid too strong of a mismatch between training and testing, the randomly selected 15% of input tokens are not always masked. 10% of the time, the token is replaced by a random token, and in another 10% of cases, the token is not replaced at all. Hence in 80% of the cases, the token is actually masked. Many NLP tasks are focused on relationships between sentences and not only words, which is not directly trained using a language modelling objective. The BERT model is also trained on a next sentence prediction objective *i.e.* two sentence are presented and the model has to decide whether or not the second sentence follows the first sentence or not. The data is arranged in such a way that this is the case 50% of the time.

4.8.2 Unsupervised Representation Learning for ASR

4.8.2.1 wav2vec and vq-wav2vec

One of the first works that used contrastive learning for ASR is wav2vec (Schneider et al., 2019). Given the raw audio sequence $x_{0:T_1-1}$ (the waveform), an encoder network consisting of multiple causal Convolutional Neural Network (CNN) layers embeds the waveform into a sequence of vectors $\mathbf{z}_{0:T_2-1}$. Kernel sizes and strides are designed in such a way that each vector $\mathbf{z}_i$ comes from 30 ms of audio and is strided at 10 ms *i.e.* $\frac{T_1}{T_2} = 160$. Another series of CNN layers creates the embedding $\boldsymbol{c}_{0:T_2-1}$ from $\mathbf{z}_{0:T_2-1}$ where $\boldsymbol{c}_i$ depends on $\mathbf{z}_{i-v:i}$. Schneider et al. (2019) used $v = 18$. The embeddings $\boldsymbol{c}_{0:T_2-1}$ are used as the input to an Acoustic Model (AM) instead of other features such as Filter Bank (FBANK) features or Mel-scale Frequency Cepstral Coefficients (MFCCs). The model *i.e.* the ANN mapping $x_{0:T_1-1}$ to $\boldsymbol{c}_{0:T_2-1}$, is trained to distinguish a sample $\mathbf{z}_{i+k}$ that is k time steps in the future from multiple distractor samples $\tilde{\mathbf{z}}$ that are drawn from a proposal distribution, by minimising the following contrastive loss for each step size $j = 1 \ldots \Theta$:

$$\mathcal{L} = \sum_{j=1}^{\Theta} \mathcal{L}_j \tag{4.14}$$

$$\mathcal{L}_j = -\sum_{i=1}^{T-j} \left(\log \sigma\left(\mathbf{z}_{i+j}^T h_j\left(\boldsymbol{c}_i\right)\right) + \lambda \mathbb{E}\left[\log \sigma\left(-\tilde{\mathbf{z}}^T h_j\left(\boldsymbol{c}_i\right)\right)\right]\right) \tag{4.15}$$

where $h_j(\cdot)$ is a separate affine transformation for each step j ($h_j(\boldsymbol{c}_i) = W_j\boldsymbol{c}_i + \boldsymbol{b}_j$). The expectation is approximated by sampling ten negative examples uniformly from the same audio sequence. Sampling from the same audio sequence gave better results than sampling from other audio sequences. In Schneider et al. (2019) $\Theta = 12$ was used *i.e.* a prediction of the future 120 ms.

In Baevski et al. (2020a) (vq-wav2vec), the goal is to learn a discrete speech representation instead of a continuous one. The embedding $\mathbf{z}_{0:T_2-1}$ is transformed into a "discrete" embedding by choosing an embedding $\tilde{\boldsymbol{z}}_i \in \mathcal{V}$ from a codebook V. This is done using the Gumbel-Softmax trick (Jang et al., 2017), which is a differentiable version of taking the argmax of the Softmax function. The same contrastive loss as in Equation (4.15) can then be used on these discrete embeddings. However, using these discrete representations resulted in a significant reduction in performance over using the continuous embeddings. But at the same time, these discrete representations allow the use of algorithms that were designed for discrete data, such as language modelling techniques. Baevski et al. (2020a) applied a BERT-model with masked prediction training to the discrete embeddings after using the contrastive loss. This gave significant improvements and outperformed using the continuous wav2vec (*i.e.* Schneider et al. (2019)).

4.8.2.2 wav2vec 2.0

Baevski et al. (2020b) (wav2vec 2.0) has a lot of similarities with Schneider et al. (2019) (wav2vec). Given the raw audio sequence $x_{0:T_1-1}$, an encoder network consisting of temporal CNN layers embeds the waveform into a sequence of vectors $\mathbf{z}_{0:T_2-1}$. Kernel sizes and strides are designed in such a way that each vector $\mathbf{z}_i$ comes from 25 ms of audio and is strided at 20 ms *i.e.* $\frac{T_1}{T_2} = 320$. Instead of using CNN-layers, Transformer layers (Vaswani et al., 2017) (see Section 2.1.7) are used to obtain the embedding $\boldsymbol{c}_{0:T_2-1}$ from $\mathbf{z}_{0:T_2-1}$. A 1D-convolutional layer, essentially a Time Delay Neural Network (TDNN)-layer is used to provide positional embeddings. The main difference is that now each $\boldsymbol{c}_i$ depends on the whole sequence $\mathbf{z}_{0:T_2-1}$. wav2vec 2.0 also uses a discretisation module as in Baevski et al. (2020a), however not for the representations but to get the targets. The embeddings $\mathbf{z}_{0:T_2-1}$ are transformed into "discrete" embeddings by choosing an embedding $\tilde{\boldsymbol{z}}_i \in V$ from a codebook V and again, this uses the Gumbel-Softmax trick. Inspired by BERT (see Section 4.8.1) during training a certain number of features in $\mathbf{z}_{0:T_2-1}$ are masked. Baevski et al. (2020b) sampled 6.5% of the indices $0 : T_2 - 1$ as starting indices and then masked the subsequent 10 vectors. This resulted in around 49% of the vectors being masked. A contrastive loss is used that more closely resembles the loss in

SimCLR (Equation (4.13)):

$$\mathcal{L}_m = -\log \frac{\exp(\text{sim}(\boldsymbol{c}_t, \mathbf{q}_t)/\kappa)}{\sum_{\hat{\mathbf{q}}\sim \boldsymbol{Q}} \exp(\text{sim}(\boldsymbol{c}_t, \hat{\mathbf{q}})/\kappa)} \tag{4.16}$$

For each $\boldsymbol{c}_t$, which is the context network output centred over the masked time-step t, the model needs to identify the true quantised latent speech representation $\mathbf{q}_t$, in a set of $K+1$ quantised candidate representations $\hat{\mathbf{q}}$, which includes $\mathbf{q}_t$ and K distractors. In addition, a regularisation loss is used that encourages diversity of the codebook entries used:

$$\mathcal{L}_d = \frac{1}{\|V\|} \sum_{v=1}^{\|V\|} p_v \; \log \; p_v \tag{4.17}$$

where p_v is the averaged Softmax distribution over the codebook entries *i.e.* the negative entropy over the probabilities of each codebook entry. Wav2Vec 2.0 based embeddings have been used as the input representation for ASR-models and for speaker-embedding models. The wav2vec 2.0 loss has also been used to obtain multi-lingual speech representations (Babu et al., 2021).

4.8.2.3 HuBERT

Hidden unit BERT (HuBERT) (Hsu et al., 2021a) was inspired by Caron et al. (2018) which was inspired by cluster-then-label approaches, while at the same time working from the results achieved by wav2vec 2.0. HuBERT uses the same encoder $x_{0:T_1-1} \to \mathbf{z}_{0:T_2-1} \to \boldsymbol{c}_{0:T_2-1}$ as wav2vec 2.0. Similar to wav2vec 2.0 $\mathbf{z}_{0:T_2-1}$ is masked by choosing a certain percentage of vectors as starting points (in Hsu et al. (2021a) 8% was used) and then masking the next 10 vectors (yielding around 65% of vectors being masked due to overlapping masks). Instead of using a contrastive loss for the masked positions, a discriminative loss is used by predicting cluster labels. As in Caron et al. (2018), K-means clustering was used. To get the first iteration of cluster labels, MFCC features are clustered, and the HuBERT model is trained on these (in Hsu et al. (2021a) 100 clusters are used in the K-Means clustering). After training using the K-Means clustering, the embeddings of one of the transformer layers are clustered, and those cluster labels are used. This second iteration significantly improved the HuBERT embeddings. WavLM (Chen et al., 2022) showed that jointly learning the masked speech prediction of HuBERT and a denoising task yields superior embeddings. HuBERT is used as the baseline semi-supervised learning algorithm in Chapter 9. This learning algorithm is presented in more depth in Section 9.1.1.

4.9 Summary

This chapter introduces semi-supervised learning, particularly for ASR. The semi-supervised learning modes, transductive and inductive semi-supervised learning, were explained. Many semi-supervised learning approaches were explained. The very popular self-training approach was explained, as well as lightly-supervised training, probabilistic-generative models and cluster-then-label approaches. Cluster-then-label approaches inspired DeepCluster, which inspired HuBERT, which is the starting point of Chapter 9. Consistency regularisation, particularly VAT is explained, which is a precursor to the cosine-distance virtual adversarial training (CD-VAT) method proposed in Chapter 6. Finally, unsupervised representation learning methods were explained, which is further explored in the methods discussed and proposed Chapter 9.

Chapter 5

Active Learning

Chapter 4 described semi-supervised learning as a method to improve the performance of machine learning algorithms trained on supervised data alone. This, in turn, would allow semisupervised learning to achieve the same performance with less labelled data. This is important in domains where labelled data is expensive to come by, but unlabelled data is much cheaper and more readily available. This could be the case for data that is used for Automatic Speech Recognition (ASR), Object Recognition or Machine Translation. Active learning starts from a similar operating point. A small amount of labelled data is available together with a much larger pool of unlabelled data. Limited by a constrained budget, the task of active learning is to choose a portion of the unlabelled data to be labelled. The data is chosen in such a way that less labelled data is needed in order to achieve the same performance in comparison to randomly choosing unlabelled data to be labelled. The machine learning model itself, or an external model, asks an external “oracle" (for example, a human annotator) for a label for the chosen unlabelled data. In effect, the active learning algorithm will only learn what it needs to in order to improve, thus reducing the overall cost of training an accurate system.

For some simple machine learning models, the unlabelled input data can actually be synthesised directly. This, however, can be awkward when the oracle is a human annotator. For example, in Lang and Baum (1992) a model directly synthesised handwritten characters which were given to a human oracle label such that later a neural network could be trained. However, many of the generated characters were unrecognisable symbols. Given that this form of active learning is likely to be impossible for speech recognition, it won’t be further discussed. The two operating points of active learning that are useful are *stream-based* and *pool-based* scenarios. In a stream-based scenario, the data comes one sample or utterance at a time, and the active learning algorithm must decide whether to query the oracle or discard the data. In a pool-based active learning scenario, it is assumed that there is a small set of labelled data $\mathcal{D}_l$ and a large *pool* of unlabelled data $\mathcal{D}_{ul}$ available. The active learning algorithm then selects a subset of the unlabelled data to be labelled by the oracle. In most research applications *pool-based* is a natural setting given the fixed data sets that are being used. In real-world applications where data-storage requirements can be a problem, stream-based approaches become more important. Samples from a "stream" in this context can be temporarily stored as a pool and a batch of it be selected for labelling as an intermediate operating point. If not otherwise noted, the following will refer to pool-based active learning. Further, active learning will be discussed with a focus on classification tasks, though active learning can also be applied to regression tasks (MacKay, 1992).

5.1 Problem Statement and Notation

As in semi-supervised learning, the available data consists of both L labelled instances

$$\mathcal{D}_l = \{(\mathbf{x}_i; \hat{\mathbf{y}}_i)\}_{i=1}^{L} \tag{5.1}$$

and U unlabelled instances

$$\mathcal{D}_{ul} = \{\mathbf{x}_i\}_{i=L+1}^{L+U} \tag{5.2}$$

where often $U \gg L$.

The task of active learning is to select a subset of the unlabelled data $S \subseteq \mathcal{D}_{ul}$ whose total labelling cost is less than the budget C. The choice of data points to be labelled uses an *acquisition function*, $f(S)$, which computes the potential usefulness of a subset of the unlabelled data for the supervised learning task. The overall active learning problem can thus be described as,

$$S^* = \arg\max_{S} f(S), \text{Cost}(S) \leq C \tag{5.3}$$

In many cases, the acquisition function examines the data points individually *i.e.* it ranks the unlabelled data points based. In this case, the overall active learning problem can be described as,

$$S^* = \arg\max_{S} \sum_{\mathbf{x} \in S} f(\mathbf{x}), \text{Cost}(S) \leq C \tag{5.4}$$

In many active learning methods, the acquisition function $f()$ depends on the labelled data set $\mathcal{D}_l$. It is important to note that clearly, the best strategy would be to directly select those data points that minimise the classification error rate (or Word Error Rate (WER)) of the machine learning model. However, for almost all machine learning algorithms, there is no direct strategy to achieve this goal. Hence, approximations are made, and measures are found that are supposed to correlate well with the error rate.

When the acquisition function considers the subset as a whole, choosing the best subset becomes an NP-hard combinatorial optimisation problem. Constrained submodular function maximisation can be used to solve this problem with constant-factor guarantees (Lin and Bilmes, 2011; Wei et al., 2014a).

5.2 Overview of active learning in ASR

In active learning for ASR, a seed model is trained on a small amount of labelled data (utterances with transcriptions). Based on an acquisition function, which is likely to involve the seed model itself, a subset of a generally large set of unlabelled utterances (pure audio) is selected to be transcribed by an external *oracle*. In an industrial application, this would involve human transcribers listening to the audio and transcribing it. In speech recognition research, a small portion of an already fully transcribed data set is taken as the initial training set,$\mathcal{D}_l$ and the rest is used as the unlabelled data set, $\mathcal{D}_{ul}$ (Fraga-Silva et al., 2015a; Hakkani-Tür et al., 2002). The oracle can then be easily simulated since the manual transcriptions exist for the unlabelled part of the data set. Using the acquisition function, multiple utterances are chosen for labelling, the model is then retrained and an additional set of utterances is chosen for transcription. The process is repeated until the “labelling budget” is exhausted. The active learning process should reduce the amount of labelling required to train the ASR system under the assumption that not all utterances are equally important to improve the model. This could be, for example, due to some utterances being very similar to each other.

5.2.1 Value of Active Learning

Accurately transcribing audio data is difficult and laborious. In many situations, it takes significantly longer than real-time to transcribe audio data. At the same time, it has been shown that modern speech recognisers perform increasingly well with the amount of data with which they have been trained (Amodei et al., 2016; Chan et al., 2021; Miao et al., 2016; Saon et al., 2017). The amount of labelled data used to train speech recognition systems in English or Mandarin is so large that it is only done for these commercially very lucrative languages. Thus, using active learning to reduce labelling efforts significantly can help to democratise speech recognition technology. Furthermore, speech recognisers trained on large amounts of data associate the same cost to mistranscribing each utterance. This can lead to an overemphasis of the speech recogniser on types of utterances that frequently occur in the training set, on which the speech recogniser might already be very good. In turn, this can lead to bad performance on rare types of utterances with, for example, rare accents. Active learning could also help to improve the poorest performing aspects of a speech recogniser, which causes a technology to be unusable for some users, rather than further improving the speech recogniser for users for which the technology is already very usable.

5.2.2 Challenges of AL in ASR

As should be apparent from Chapter 3, ASR systems consist of multiple independently trained components. Mainly an ASR system is divided into an acoustic model and a language model. The acoustic model requires both audio and text data to be trained, whilst the language model uses only text data. An utterance-transcription pair that might improve the acoustic model might not necessarily improve the language model and vice versa. At the same time, the acoustic model also consists of independent components (decision tree, HMM, ANN) and the training is often done in multiple stages, e.g. HMM-GMM models, alignments, ANN CE training, lattice generation, and MPE training. In mini-batch adaptive active learning, this yields the problem that if, for each iteration, each component is retrained using all stages, the process is very expensive whilst not redoing each stage (e.g. using just generating new lattices for the new utterances and doing MPE training) might yield an inferior model for the active learning. There is likely to be a trade-off which is helped by the fact that we are interested in the subset of data chosen and not the model used to obtain it. A model trained in a streamlined manner might select the same subset of data to be labelled as a model trained in the most complicated fashion. Whilst there is a near infinite amount of unlabelled audio available, most of the audio is likely to be irrelevant to the target domain of the speech recogniser. In research settings, it is common for all the data to be in-domain.

5.2.3 Labelling Cost in ASR

Many factors influence the transcription cost (Glenn et al., 2010). One of the main drivers is the desired quality and richness of the transcriptions. Table 5.1 outlines the different options for how carefully the transcriptions can be done by the transcriber. The partial words that are not used in WER calculations (see Section 3.6) can be included or ignored. Filler words such as ‘uh’ or ‘um’ can be included. Proper nouns can be explicitly marked by the transcriber or left in lower case form. Even more detail can be added such as explicitly marking if a word was mispronounced or if a non-standard or foreign word (code-switch) was used. Furthermore, any verification passes significantly increase the labelling cost.

	Quickest	→		Most Careful
Segmentation	Automatic	Auto w/ verification	Manual	Manual w/ verification
Completeness	Content words	Add partial words, disfluencies	Add partial words, disfluencies	Add verification pass
Filled Pauses	Optional	Incomplete	Exhaustive	Exhaustive w/ verification
Disfluencies	None	Incomplete	Exhaustive	Exhaustive w/ verification
Transcriber Uncertainty	Flag and skip	Flag and best guess	Flag and best guess	Flagged best guess w/ verification
Feature Marking	None	Minimal	Full	Accurate, complete w/ correction
Speaker, Background Noise	None	Minimal	Exhaustive	Exhaustive w/ verification
Manual Passes	1	1-2	2-3	4+
Approx. Cost (x Real Time)	5x	15x	25 x	50 x

Table 5.1 Overview of transcription approaches, from quickest to most careful (Cieri and Strassel, 2007).

The Fisher corpus (Cieri et al., 2004b) is a speech recognition corpus that was explicitly collected using a lower-cost protocol in order to collect much more transcribed data than ever before. Towards this effort, the Quick Transcription Specifications were developed. The transcription protocol relies on automatic segmentation of the audio into utterances. There was only a single pass over the audio in order to create the transcriptions, and no special effort was used to provide capitalisation (*e.g.* to mark proper nouns) or mark any special features (*e.g.* mispronunciation). Kimball et al. (2004) showed that using the quick transcriptions over more accurate transcriptions yields an increase in WER that is small enough to justify this protocol.

Lower transcription quality at a low cost can also be achieved by using non-professional transcribers via services such as Amazon's Mechanical Turk (Audhkhasi et al., 2012; Novotney and Callison-Burch, 2010; Parent and Eskenazi, 2011). Given that the quality of these transcriptions is likely lower than that of professionals, multiple transcriptions can be combined to yield improved transcriptions Evanini et al. (2010); van Dalen et al. (2015). Whilst this does not reduce the real-time factor, the wages paid to the non-professionals are much lower. van Dalen et al. (2015) reports a reduction in cost by a factor of 10.

Another low-cost solution is closed captions, which can be used together with lightly-supervised training (see Section 4.5). Closed captions, as broadcast on television, are typically produced using a process called respeaking. Respeaking, as carried out by a respeaker, is the process of repeating what is heard into speech recognition software, which is trained to that specific individual's voice and pronunciation *i.e.* a speaker-specific ASR system. The respeaker essentially listens and speaks at the same time. This ASR system and associated software use the audio input from the respeaker to generate the caption text. The real-time factor of respeaking can be very close to 1 but the WER of the transcription is around 33% (Bell et al., 2015), of which 26% is due to deletions and 7% is due to insertions or substitutions. This is because the respeaker does not need to produce a literal transcript at all times and may summarise portions of the audio.

Previous work on active learning in ASR usually treats the total number of hours of a data set or the number of utterances as the labelling cost. Thus, the labelling budget is reached when a certain number of hours of data is selected or if a certain number of utterances is selected. This is not representative of real-world labelling costs. For certain utterances in, say a dialogue, it can be necessary to listen to neighbouring utterances in order to determine the correct transcription. Thus, it would likely be cheaper to transcribe entire conversations rather than individual utterances. This motivates the work in Section 8.2 where the reduced cost of transcribing neighbouring utterances is taken into account.

However, there are many other aspects that can influence the cost of accurately transcribing a specific utterance. Speech varies greatly among speakers due to accents, dialects, speech impediments, and personal speaking styles. Utterances of speakers with speech impediments or a more difficult-to-recognise speaking style might increase the number of passes needed to transcribe the speech accurately. Further, if the transcriber is unfamiliar with the accent or dialect of the speaker, this might also increase the cost. Background noise and overlapping speech can also be factors that influence transcription cost.

5.3 Optimisation of Acquisition Functions

Beyond the above-mentioned active learning settings, stream-based and pool-based active learning, which describe how the input data arrives it is also important to consider how the labels arrive. In *batch active learning*, a solitary round of data selection occurs, with chosen data points being labelled without any insight into the forthcoming labels. *Adaptive active learning* involves multiple data selection rounds, with each round selecting a single data point, the label of which informs future rounds' data point selections. As a blend of both, *mini-batch adaptive active learning* entails selecting a mini-batch of data points to be labelled in each round, which subsequently influences the selection of subsequent mini-batches.

It is obvious that adaptive active learning should always perform at least as well as batch active learning. However, at the same time, it is also a more expensive procedure. Given a newly labelled data point, the model used for the data selection has to be re-trained, and the acquisition function has to be re-computed for the unlabelled data points. In many cases, adaptive active learning will perform much better than batch active learning. Consider a confidence-based approach. If the sample on which the model is the least confident is included in the unlabelled data set multiple times, then batch active learning will ask for a label for it multiple times. In adaptive active learning, the first round will result in labelling this data point on which is model is likely to be confident after training. Therefore the same data point will not be asked to be labelled in future rounds.

5.4 Acquisition Functions

This section discusses various choices for the acquisition function $f(\mathbf{x}_i)$. A higher value for $f(\mathbf{x}_i)$ indicates a higher potential usefulness for the supervised learning task.

5.4.1 Confidence-Based Methods

The most commonly used acquisition function is based on *confidence* or, in turn, *uncertainty*. This framework gives a high score to data points on which the model trained on $\mathcal{D}_l$ has a low confidence (Lewis and Catlett, 1994). Thus, these functions can only be used with probabilistic models. The acquisition function is given by

$$f(\mathbf{x}_i) = 1 - P_\theta\left(y^\star|\mathbf{x}_i\right), \quad y^\star = \arg\max_y P_\theta\left(y|\mathbf{x}_i\right) \tag{5.5}$$

Equation (5.5) could be interpreted as the model's belief that it will mislabel $\mathbf{x}_i$. Similarly, the output entropy can also be used as a measure of how uncertain the model is about its own prediction. For binary classification both the least-confident and the entropy approach are equivalent. As described in Section 2, there are different types of uncertainty: epistemic, aleatoric and predictive uncertainty. To improve the model parameters, it is desirable to select samples that decrease the uncertainty about the parameters. One example of such an acquisition function is called Bayesian Active Learning by Disagreement (BALD) (Houlsby et al., 2011):

$$f(\mathbf{x}_i) = \mathbb{H}_{P(y|\mathbf{x}_i,\mathcal{D}_l)}[y] - \mathbb{E}_{p(\theta|\mathcal{D}_l)}[\mathbb{H}[y|\mathbf{x}_i,\theta]] \tag{5.6}$$

where $P(y|\mathbf{x}_i,\mathcal{D}_l)$ is the predictive distribution and $p(\theta|\mathcal{D}_l)$ described in Section 2.4.1.

Points that maximise this acquisition function are points on which the model is uncertain on average, but the probable model parameters produce disagreeing predictions each with high certainty. This acquisition function needs a posterior for the parameters $p(\theta|\mathcal{D}_l)$. In a deep neural network, similar to Section 2.1, this can be solved using approximate variational inference and Monte Carlo sampling (Gal et al., 2017b).

A large proportion of work done on active learning for ASR uses an acquisition function that selects utterances on which the model is most uncertain. Usually, this is done using confidence score approaches (Hakkani-Tür et al., 2002; Kamm and Meyer, 2002, 2003; Riccardi and Hakkani-Tur, 2005; Yu et al., 2010b, 2007). Otherwise, uncertainty has been measured using lattice entropy (Yu et al., 2010a) or using the entropy of the confidence scores of the n-best list (Itoh et al., 2012).

5.4.2 Query-by-Committee

Another approach is called Query-By-Committee (QBC) (Seung et al., 1992). The QBC approach involves creating an ensemble of classifiers (here called the *committee*) trained using the same labelled data set $\mathcal{D}_l$. Each of the classifiers has to have a different structure or has to be trained in a different way such that the classifiers give different hypotheses. Based on the hypotheses given by each classifier, the unlabelled data point considered to be most informative will be the one for which the classifiers disagree the most. This can be seen to be similar to BALD, but can be used without the posterior probability for any parameters (or an approximation thereof). For QBC algorithms, it is important to find a good measure for the disagreement between different hypotheses. For classification, an option is the voting entropy (Dagan and Engelson, 1995):

$$f(\mathbf{x}_i) = -\sum_{k=1}^{K} \frac{\mathcal{K}(y_k|\mathbf{x}_i)}{J} \log \frac{\mathcal{K}(y_k|\mathbf{x}_i)}{J} \tag{5.7}$$

where $\mathcal{K}(y_k|\mathbf{x}_i)$ is the number of classifiers that chose label y_k for input $\mathbf{x}_i$ and Jis the total number of classifiers in the ensemble. $\frac{\mathcal{K}(y_k|\mathbf{x}_i)}{J}$ can be seen as the output distribution of the ensemble (assuming each classifier outputs only a class and not a distribution itself). This leads to $f(\mathbf{x}_i)$ being the entropy of this distribution. Another method proposed by McCallum and Nigam (1998) is the average Kullback-Leibler divergence (KL-divergence) between each classifier and the average of the classifiers. For this method, any other divergence measure can be used, such as the Jensen-Shannon divergence (Melville et al., 2005).

The current models used for speech recognition can output a distribution over possible hypotheses (usually in the form of a lattice or an n-best list). The uncertainty over different hypotheses can only capture the aleatoric uncertainty. A way to capture the model uncertainty would be to use a model that can output a distribution over distributions. The entropy of that distribution over distributions would model how uncertain the model is over its own prediction (here the distribution over hypotheses) and therefore over its own parameter setting. The work in Hamanaka et al. (2010) used an approach where the disagreement between different hypotheses is considered. Here, ten models were each trained on 10% of the training data and then a disagreement measure based on the 1-best of the ten models. Training on only 10% of the data means that each of the models is by far weaker than if trained on the full set.

5.4.3 Expected Model Change

Another general active learning framework uses a decision-theoretic approach, selecting the instance that would impart the greatest change to the current model *if we knew its label*. An example of this is the expected gradient length (EGL) approach which can be used for models trained with gradient descent (Settles et al., 2008). In gradient-based optimisation the “change" imparted to the model can be measured by the length of the training gradient for the cost function C. When using EGL, the learner selects unlabelled data which, if labelled and added to $\mathcal{D}_l$ would result in the new training gradient of the largest magnitude.

$$f(\mathbf{x}_i) = \sum_{k=1}^{K} P_\theta(y = k|\mathbf{x}_i) \left\|\nabla_\theta C \left(\mathcal{D}_l \cup (y = k, \mathbf{x}_i)\right)\right\| \tag{5.8}$$

Given that the model is likely to be trained sufficiently such that the gradient for the training set is close to zero, the acquisition function is well approximated by just considering the new sample.

$$f(\mathbf{x}_i) \approx \sum_{k=1}^{K} P_\theta(y = k|\mathbf{x}_i) \left\|\nabla_\theta C \left(y = k, \mathbf{x}_i\right)\right\| \tag{5.9}$$

For speech recognition, this method was used by Huang et al. (2016) with a Connectionist Temporal Classification (CTC) loss.

5.4.4 Representativeness-Based Methods

The methods discussed above mostly consider the future performance of the learner on a specific unlabelled data point. They do not consider how labelling this data point might affect future performance on other samples. Samples for which the model has low confidence could simply be noisy garbage samples, for which the model should be uncertain. This motivates approaches that try to sample instances that are not only informative but also *representative*. In most cases, a combination of informativeness and representativeness is used. First introduced by McCallum and Nigam (1998), a QBC approach is de-weighted by a distance measure between different samples, which was applied to text classification. Settles and Craven (2008) introduces the *information density* framework in which the acquisition function tries to find samples that maximise the product of the informativeness and the average similarity of the sample with the rest of the unlabelled data set:

$$f(\mathbf{x}_i) = f'(\mathbf{x}_i) \cdot \left(\frac{1}{U} \sum_{j=L+1}^{U+L} \mathtt{sim}(\mathbf{x}_j, \mathbf{x}_i) \right)^{\beta} \tag{5.10}$$

where, $f'(\cdot)$ can be any of the previously defined acquisition functions, $\mathtt{sim}(\cdot,\cdot)$ is any similarity measure between two samples and β is a coefficient that weights the two criteria.

Representativeness methods can also be applied on their own without any other acquisition function. Such that Equation (5.10) is stripped down to:

$$f(\mathbf{x}_i) = \sum_{j=L+1}^{U+L} \mathtt{sim}(\mathbf{x}_j, \mathbf{x}_i) \tag{5.11}$$

In active learning for ASR, utterances selected based on any acquisition function can be outlier utterances that do not contribute towards improving the acoustic model. These could be noise or garbage utterances. This can be overcome by including a similarity measure into the acquisition function. The similarity measure gives a score about how similar the utterance is to the rest of the data set and thus helps to result in a representative subset. This was first done by Yu et al. (2010a) and then by Itoh et al. (2012). Itoh et al. (2012) applied the information density framework by Settles and Craven (2008) (Equation (5.10)) to ASR. The function $f'(\mathbf{x})$ was the entropy of the confidence scores of the N-best list. The similarity measure was the cosine distance between vectors containing term frequency inverse document frequency (TF-IDF) scores for each term, where a term was a phone-n-gram in the transcription obtained by decoding using the seed model. A purely representativeness approach was used by Lin and Bilmes (2009), without any seed data. The similarity measure was the L_1-distance between Fisher scores (Bilmes, 2008; Jaakola and Haussler, 1999) with respect to the parameters of a Hidden Markov Model (HMM) trained on $\mathcal{D}_{ul}$. The objective function was the facility location function, chosen for its submodularity. Wei et al. (2014b) used a modification to the gapped string kernel (Rousu and Shawe-Taylor, 2007), in the form given by Wei et al.

(2013), for the facility location function. The tokenisation of the utterances, needed for the kernel, used a 40 state HMM with 64 Gaussians for each state which was trained on the unlabelled data and aligning using the Viterbi algorithm. Both Lin and Bilmes (2009) and Wei et al. (2013) performed experiments on the TIMIT corpus and reported the facility location function to outperform random selection. Syed et al. (2016) used the facility location function under the IARPA Babel evaluation framework for Swahili. A 2 hour subset is selected with a 1 hour $\mathcal{D}_l$ from a 30 hour supervised data set $\mathcal{D}_{ul}$. However, it was reported that the facility location function performed slightly worse than random selection. Syed et al. (2016), also interpolates the facility reward function with an approach that encourages diversity between selected utterances (See Section 5.4.5). The data set is partitioned and a score is added that scales linearly with the number of clusters that intersect with the selected subset.

A disadvantage of these methods is that the similarity measures are usually computed for every pair of utterances. These approaches are, therefore, too computationally complex for very large sets of untranscribed utterances and cannot be used in stream-based active learning.

5.4.5 Coverage Based Methods

An issue with the representativeness approach is that the larger the training data set, the more redundancy it will contain. Thus, another approach is to try to make sure that the subset chosen to be labelled is as diverse as possible and covers different features. Fujii et al. (1998) combined the approach of Section 5.4.4 by selecting data points that are the least similar to the currently labelled samples but the most similar to unlabelled instances. Another approach, also based on similarity measures, was used by Wei et al. (2014a). It is an acquisition function that operates on the entire selected subset. First, the unlabelled data set is partitioned into clusters F_k.

$$\bigcup_{k=1}^{K} F_k = \mathcal{D}_{ul} \tag{5.12}$$

All F_k are disjoint, and the acquisition function is defined by:

$$f(S) = \sum_{k=1}^{K} \sqrt{\sum_{\mathbf{x}_j \in \left\{F_k \bigcap S\right\}} \left(\frac{1}{U} \sum_{m=L+1}^{U} \mathtt{sim}\left(\mathbf{x}_m, \mathbf{x_j}\right) \right)} \tag{5.13}$$

where, $\mathtt{sim}$ is any similarity metric. Given that the square root function is a monotone non-decreasing concave function, the contribution to the acquisition function is larger if a sample is chosen from a cluster that has not had many of its elements chosen. This function is termed the *diversity reward function* and is sub-modular, meaning it can be optimised in a greedy fashion with constant factor guarantees.

In speech recognition coverage based methods have been used by Wei et al. (2014a),Chen et al. (2015) and Fraga-Silva et al. (2015a,b). Chen et al. (2015) showed that coverage-based active learning can reduce Out-of-Vocabulary (OOV) counts.

5.5 Summary

This chapter introduced active learning, particularly for ASR. The value and the challenges of active learning for ASR were discussed and a detailed account of the labelling cost for ASR was given. The labelling cost in ASR will be a major part of Chapter 8. Various acquisition functions were outlined. These include confidence-based methods which serve as a baseline for the experiments in Chapter 7 and are used for the experiments in Chapter 8. Representative-based methods were also discussed, of which the information density framework will be used in Chapter 7 to balance the approach that is proposed in Chapter 7. Further, coverage-based methods were discussed. Whilst they are not used in this thesis, the methods proposed in Chapter 9 provide novel methods for tokenising an input audio sequence in a semi-supervised way. Such tokenisation could be used in the future for coverage-based methods.

Chapter 6

Consistency Regularisation for Speech Recognition and Speaker Verification

Up until the start of this work, the majority of the work that had been done within semi-supervised learning for Automatic Speech Recognition (ASR) was related to reducing the entropy on all data points (see methods in Section 4.4): either directly or by training on the predictions of the seed ASR system that was trained on the initial training set. Numerous publications within this framework were focused on how to filter out utterance-transcription pairs that might hinder training if added to the training set. One research direction that had not been examined yet was the principle of consistency regularisation (see Section 4.7).

Consistency regularisation is a form of regularisation where multiple versions (after applying data augmentation techniques) of the same unlabelled data point are passed through the same classifier. The auxiliary cost function is then the disagreement between these predictions. The difference in prediction can be defined by any distance measure between distributions. Usually, the Kullback-Leibler divergence (KL-divergence) is used, but also the squared distance between outputs or activations can be used.

In nature, the outputs of most systems are smooth with respect to (w.r.t) spatial and temporal inputs (Wahba, 1990). Prior studies have confirmed that smoothing the output distribution of a classifier (*i.e.*, encouraging the classifier to output similar distributions) against perturbations of the input can improve its generalisation performance in semi-supervised learning (Laine and Aila, 2017; Luo et al., 2018; Miyato et al., 2019; Sajjadi et al., 2016). Laine and Aila (2017) and Sajjadi et al. (2016) studied Gaussian noise and dropout as perturbations, whilst Luo et al. (2018) and Miyato et al. (2019) also studied adversarial perturbations. In the standard version of virtual adversarial training (VAT) (Section 4.7.1; (Miyato et al., 2019)) (the efficacy of which has been verified by Oliver et al. (2018) and Zhai et al. (2019)) an additive loss is introduced, which tries to smooth the categorical output distribution (measured by the KL-divergence) around every data point that lies on the data manifold. In Section 6.1, VAT is presented in more detail by expanding upon the introduction given in Section 4.7.1.

In Section 6.2, VAT is examined for the training of Hidden Markov Model (HMM)-Deep Neural Network (DNN) hybrid ASR systems. These experiments were not very successful. It is examined why for hybrid HMM-DNN models trained using Cross Entropy (CE)-training, this form of training is not effective.

In the second part of this chapter, a new form of VAT is developed, named cosine-distance virtual adversarial training (CD-VAT). CD-VAT allows VAT to be used to train semi-supervised speaker embeddings that can be used for tasks such as speaker verification where the set of speakers seen at test-time is different from the speakers seen during training. It is presented and evaluated in Section 6.3. At the time of its publication (Kreyssig and Woodland, 2020), this algorithm was the first method for semi-supervised method for speaker embedding training. Earlier work by Stafylakis et al. (2019) also developed semi-supervised speaker embeddings, but with the help of a fully trained speech recogniser train on additional supervised ASR data.

6.1 Virtual Adversarial Training

As introduced in Section 4.7.1, the cost function that is used in VAT is the scaled addition of the supervised cost and the VAT cost,

$$\mathcal{C}\left(\mathcal{D}_l, \mathcal{D}_{ul}, \theta\right) = \mathcal{C}_{\mathrm{S}}(\mathcal{D}_l, \theta) + \alpha \mathcal{R}_{\text{VAT}}(\mathcal{D}_l, \mathcal{D}_{ul}, \theta) \tag{6.1}$$

where the regularisation loss $\mathcal{R}_{\text{VAT}}(\mathcal{D}_l, \mathcal{D}_{ul}, \theta)$ is given by,

$$\mathcal{R}_{\text{VAT}}(\mathcal{D}_l, \mathcal{D}_{ul}, \theta) = \frac{1}{L+U} \sum_{\mathbf{x} \in \mathcal{D}_l, \mathcal{D}_{ul}} \text{LDS}(\mathbf{x}, \theta) \tag{4.9}$$

$$\text{LDS}(\mathbf{x}, \theta) = D[P_{\theta'}\left(y \mid \mathbf{x}\right), P_{\theta}\left(y \mid \mathbf{x} + \mathbf{r}_{\text{VAT}}\right)]\bigg|_{\theta'=\theta} \tag{4.10}$$

where $\text{LDS}(\mathbf{x}, \theta)$ is termed the local-distribution smoothness as it measures the disturbance to the output distribution due to the perturbation $\mathbf{r}_{\text{VAT}}$ as measured by the distance $D[\cdot, \cdot]$. The separation between θ and θ' is intentionally chosen to emphasise that, while the gradients are computed with respect to θ they are only computed for the right-hand-side of the distance function. θ' represents a current, fixed snapshot of the parameters. $\mathbf{r}_{\text{VAT}}$ is defined by,

$$\mathbf{r}_{\text{VAT}} = \operatorname*{arg\,max}_{\mathbf{r}; \|\mathbf{r}\| \leq \epsilon} D[P_{\theta}\left(y \mid \mathbf{x}\right), P_{\theta}\left(y \mid \mathbf{x} + \mathbf{r}\right)] \tag{4.11}$$

i.e. $\mathbf{r}_{\text{VAT}}$ is the input perturbation of magnitude ϵ that disturbs the output distribution the most as defined by the distance $D[\cdot,\cdot]$, which for the standard VAT algorithm is the KL-divergence. The VAT loss $\mathcal{R}_{\text{VAT}}(\mathcal{D}_l, \mathcal{D}_{ul}, \theta)$ is the sum of the per-data-point losses $\text{LDS}(\mathbf{x}, \theta)$.

6.1.1 Approximation of $\mathbf{r}_{\text{VAT}}$ with Taylor series and finite differences

Given the adversarial perturbation $\mathbf{r}_{\text{VAT}}$, the optimisation of the combined cost $\mathcal{C}\left(\mathcal{D}_l, \mathcal{D}_{ul}, \theta\right)$ is straightforward (Equation (6.13) can be used) because the gradients of $\text{LDS}(\mathbf{x}, \theta)$ w.r.t θ are well defined. In this section, a method for approximately finding $\mathbf{r}_{\text{VAT}}$ is described. For simplicity, let:

$$D(\mathbf{r}, \mathbf{x}, \theta) = D[P_{\theta'}\left(y \mid \mathbf{x}\right), P_{\theta}\left(y \mid \mathbf{x} + \mathbf{r}\right)] \tag{6.2}$$

$D(\mathbf{r}, \mathbf{x}, \theta)$ has a minimum of zero at $\mathbf{r} = \mathbf{0}$. Therefore, the gradient w.r.t $\mathbf{r}$ is also zero at $\mathbf{r} = \mathbf{0}$. Therefore, the truncated Taylor series of $D(\mathbf{r}, \mathbf{x}, \theta)$ is given by:

$$D(\mathbf{r}, \mathbf{x}, \theta) \approx \frac{1}{2}\mathbf{r}^T \mathbf{H}(\mathbf{x}, \theta)\, \mathbf{r} \tag{6.3}$$

where $\mathbf{H}(\mathbf{x}, \theta) = \nabla\nabla_{\mathbf{r}} D(\mathbf{r}, \mathbf{x}, \theta)\,|_{\mathbf{r}=\mathbf{0}}$. For simplicity, let $\mathbf{H} = \mathbf{H}(\mathbf{x}, \theta)$. Under the approximation in Equation (6.3), $\mathbf{r}_{\text{VAT}}$ emerges as the dominant eigenvector $\mathbf{u}(\mathbf{x}, \theta)$ of $\mathbf{H}$ with magnitude ϵ (see constraint from Equation (6.21)). This shows that VAT in effect penalises $\lambda_1(\mathbf{r}, \mathbf{x}, \theta)$, the largest eigenvalue of $\mathbf{H}$:

$$D(\mathbf{r}, \mathbf{x}, \theta) \approx \frac{1}{2}\epsilon^2 \lambda_1(\mathbf{r}, \mathbf{x}, \theta) \tag{6.4}$$

The dominant eigenvector, $\mathbf{u}(\mathbf{x}, \theta)$, of $\mathbf{H}$ can be found using the standard power iteration method (Kreyszig et al., 2011; von Mises and Pollaczek-Geiringer, 1929) combined with finite differences. Let $\mathbf{v}_0$ be a randomly sampled vector that is not orthogonal to $\mathbf{u}(\mathbf{x}, \theta)$. Then the iterative calculation of

$$\mathbf{v}_{i+1} \leftarrow \overline{\mathbf{H}\mathbf{v}_i} \tag{6.5}$$

causes $\mathbf{v}_i$ to converge to $\mathbf{u}$. The Hessian-vector product, $\mathbf{H}\mathbf{v}_i$, can be approximated based on finite differences:

$$\nabla_{\mathbf{r}} D(\mathbf{r}, \mathbf{x}, \theta)\,|_{\mathbf{r}=\zeta\mathbf{v}_i} \approx \nabla_{\mathbf{r}} D(\mathbf{r}, \mathbf{x}, \theta)\,|_{\mathbf{r}=\mathbf{0}} + \zeta\mathbf{H}\mathbf{v}_i \tag{6.6}$$

$$\mathbf{H}\mathbf{v}_i \approx \tfrac{1}{\zeta}\nabla_{\mathbf{r}} D(\mathbf{r}, \mathbf{x}, \theta)\,|_{\mathbf{r}=\zeta\mathbf{v}_i} \tag{6.7}$$

Therefore, to obtain $\mathbf{r}_{\text{VAT}}$ the following iterative procedure can be used:

$$\mathbf{r}_{\text{VAT}} \approx \epsilon \cdot \mathbf{v}_I \tag{6.8}$$

$$\mathbf{v}_{i+1} = \frac{\mathbf{g}_{i+1}}{\|\mathbf{g}_{i+1}\|} \tag{6.9}$$

$$\mathbf{g}_{i+1} = \nabla_{\mathbf{r}} D(\mathbf{r}, \mathbf{x}, \theta) \,|_{\mathbf{r}=\zeta\mathbf{v}_i} \tag{6.10}$$

where $\mathbf{v}_I$ is the approximation of $\mathbf{u}(\mathbf{x}, \theta)$ after I power iterations and $\mathbf{v}_0$ is sampled uniformly on the unit-sphere. The value of ζ should be as small as possible to get the best estimate of $\mathbf{H}\mathbf{v}_i$, but large enough not to cause numerical issues. In the experiments in this chapter, ζ was set to values that were equivalent to a norm of 10e-6 per feature vector. This method of computing the solution to a second-order equation is only possible because the first-order gradient is zero. In cases where the gradient is not zero more complicated methods are needed (Pearlmutter, 1994).

The value of $\mathbf{g}_{i+1} = \nabla_{\mathbf{r}} D(\mathbf{r}, \mathbf{x}, \theta) \,|_{\mathbf{r}=\zeta\mathbf{v}_i}$ can be calculated using the standard backpropagation algorithm. Here, the gradient with respect to $\mathbf{r}$ is the same as the gradient with respect to the input $\mathbf{x}$,

$$\mathbf{g}_{i+1} = \left.\frac{\partial D(\mathbf{r}, \mathbf{x}, \theta)}{\partial \mathbf{r}}\right|_{\mathbf{r}=\zeta\mathbf{v}_i} \tag{6.11}$$

$$= \left.\frac{\partial \mathbf{z}}{\partial \mathbf{r}} \frac{\partial D(\mathbf{r}, \mathbf{x}, \theta)}{\partial \mathbf{z}}\right|_{\mathbf{r}=\zeta\mathbf{v}_i} \tag{6.12}$$

where $\mathbf{z}$ are the pre-softmax activations. The pre-multiplication with $\frac{\partial \mathbf{z}}{\partial \mathbf{r}}$ in Eqn. (6.12) is equivalent to the standard back-propagation algorithm. When $D[\cdot, \cdot]$ is the KL-divergence, then the gradient with respect to the pre-softmax activations $\mathbf{z}$ is given by,

$$\left[\left.\frac{\partial D(\mathbf{r}, \mathbf{x}, \theta)}{\partial \mathbf{z}}\right|_{\mathbf{r}=\zeta\mathbf{v}_i}\right]_k = P_{\theta'}\left(y = k \mid \mathbf{x}\right) - P_{\theta}\left(y = k \mid \mathbf{x} + \zeta\mathbf{v}_i\right) \tag{6.13}$$

similar to the CE-loss or the Mean Squared Error (MSE)-loss (see Equation (2.31)).

To summarise, to obtain $\mathbf{r}_{\text{VAT}}$ the required calculations are:

- a forward pass to get $P_{\theta'}\left(y = i \mid \mathbf{x}\right)$ for Equation (6.13)
- then for each power iteration:
  - a forward pass to get $P_{\theta}\left(y = i \mid \mathbf{x} + \zeta\mathbf{v}_i\right)$ for Equation (6.13)
  - a backward pass to get $\nabla_{\mathbf{r}} D(\mathbf{r}, \mathbf{x}, \theta) \,|_{\mathbf{r}=\zeta\mathbf{v}_i}$ for Equation (6.10)

Initial experiments suggested that $I = 1$ is sufficient for approximating $\mathbf{r}_{\text{VAT}}$, such that $\mathbf{r}_{\text{VAT}}$ does not change significantly for further iterations as measured by the dot-product of consecutive $\mathbf{v}_i$. This single power iteration, however, increases $D(\mathbf{r}, \mathbf{x}, \theta)$ by more than a factor of 10^4 in comparison to just using $\mathbf{v}_0$ *i.e.* the robustness of the output distribution to the simple normalised Gaussian noise $\mathbf{v}_0$ is far larger than to the adversarial noise $\mathbf{r}_{\text{VAT}}$.

Equation (6.4) can be used to check how accurate the Taylor approximation is. $\lambda_1(\mathbf{r}, \mathbf{x}, \theta)$ can be approximated as the ratio of the noise input $\zeta \cdot \mathbf{v}_i$ and the resulting gradient $\mathbf{g}_{i+1}$.

$$\lambda_1(\mathbf{r}, \mathbf{x}, \theta) = \frac{||\mathbf{g}_{i+1}||}{||\zeta \cdot \mathbf{v}_i||} \tag{6.14}$$

This was found to be very accurate when compared to directly calculating the KL-divergence.

Given the fact that $\mathbf{r}_{\text{VAT}}$ is computed based using gradients w.r.t. to input, VAT is only possible for models for which such gradients can be computed.

6.2 VAT for Speech Recognition

In the experiments within this section, the KL-divergence was used for $D[\cdot, \cdot]$ as done by Miyato et al. (2019). The supervised cost $\mathcal{C}_{\text{S}}(\mathcal{D}_l, \theta)$ is given by the CE-loss for training hybrid HMM-DNN ASR models (see Section 3.9.2). The local-distribution smoothness $\text{LDS}(\mathbf{x}, \theta)$ is also calculated at the frame level *i.e.* y is the physical HMM state s given by Viterbi alignment.

6.2.1 Experimental Setup

The initial experiments used the TIMIT data set (Garofolo et al., 1993) with the SA utterances removed, leaving 5040 utterances. Of these, the official training set of 3696 utterances was used for training. Of these utterances, 10% were sampled at random to form a validation set, and the rest were kept as the training set. This training set was further split into a supervised data set containing 10% of the data and an unsupervised data set containing the remaining 90% of the data. TIMIT was used in order to provide a setup in which VAT would be most likely to improve the model. This is because the results will show that VAT does not improve the model due to label noise. TIMIT already has a manual phone-level transcription instead of just a word-level transcription, therefore TIMIT is likely to have less label noise than data sets that have a world-level transcription.

All experiments were conducted with an extended version of HTK 3.5 within which VAT was implemented. A 40dim log-Mel Filter Bank (FBANK) analysis was used without any delta coefficients. Delta coefficients were not used because the input perturbation that VAT would generate would not match how delta coefficients are calculated. These inputs were normalised at the utterance level for mean and globally for the variance. All models were trained using the cross-entropy criterion and frame-level shuffling was used.

A decision tree with 854 clustered triphone tied-states along with GMM-HMM/DNN-HMM system training alignments were used. The NewBob+ learning rate scheduler was used to train all models. Weight-decay (see Section 2.3.1) was used. The DNN was a Time Delay Neural Network (TDNN). The TDNN-layers have the following input contexts: [-2,+2], followed by {-1,+2}, followed by {- 3,+3}, followed by {0}). All TDNN-layers have a size of 512.

The following experiments were run for analysis purposes. To obtain the decision tree and the alignments, the entire training data was used and not only the supervised data set was used. The reason for this is to use the best possible phone labels that could be obtained (for the supervised subset). The experiments will indicate that it is the inherent label noise that exists for CE-training of HMM-DNN models that means that VAT cannot be used in this scenario. Using the entire data set to obtain the alignments gives VAT the best possible chance.

All VAT related parameters were extensively tuned before coming to the negative conclusion regarding VAT for CE-training of HMM-DNN systems.

6.2.2 Experimental Results

α	Label Smoothing	VAT-Cost	Train. Acc.	Val. Acc.
1	✗	37888	71.53	58.90
5	✗	10288	55.15	50.11
10	✗	5738	47.30	45.01
20	✗	2617	28.79	29.08
1	✓	28675	68.77	58.21
5	✓	8314	55.53	50.28
10	✓	4656	47.35	45.06
20	✓	2518	40.66	39.19

Table 6.1 Effect of increasing the weight of the VAT-cost α, and of using label smoothing (✓) or not (✗). A label smoothing epsilon of 0.1 was used. Results are calculated for the TIMIT data set. Accuracies are frame-level classification accuracies.

For these experiments, $\mathbf{r}_{\text{VAT}}$ does not change significantly after the first iteration, which was measured by computing the normalised dot-product between successive perturbation vectors. This dot product was above 0.9. Hence, $I = 1$ power iterations was used.

Table 6.1 shows the results of varying α, *i.e.* the weight of the VAT-cost, and using label-smoothing. Label smoothing Szegedy et al. (2016) replaces the one-hot label for training with epsilon for the target and with epsilon divided by K for the other classes. It shows that increasing α decreases the VAT-loss of the trained model, however, it also decreases the training and validation accuracies. This shows that it is difficult for the model to jointly optimise the VAT-loss and the cross-entropy loss. It is likely that this is because VAT assumes that there are low density regions without class overlap. If there is class overlap then the CE loss tries to draw many decision boundaries in high-density areas. This is validated in Table 6.1. For high values of α, the model is unable to learn the phone-classification task. However, it learns better when label-smoothing is used. Label-smoothing does not penalise the model too much for not perfectly classifying all data points and hence for portions of the data where the VAT-loss attempts to smooth the distribution w.r.t. the input the model likely does not try to draw the decision boundaries around individual data points.

In frame-level cross-entropy training the labels are individual HMM states derived using forced Viterbi alignments. These labels are not “correct” labels, as it is impossible to define a “perfect” state-label for an individual frame. This is the cause of the inherent label noise, which then, as shown, impedes the CE-training. To illustrate this, if there would be a frame for which there is a correct label and another few frames later there is another frame for which there is also a correct label, due to the nature of speech there will be a continuous transition between the speech frames meaning that there are no “low-density" regions in the input space. The CE loss will train the model to be confident on those in-between frames, while at the same time the VAT-loss will try to prevent this.

Since then, Wang et al. (2020a) published results on using VAT for Attention-Based Encoder-Decoder (AED) models for speech recognition. Here, VAT was only used as a regulariser and not as a form of semi-supervised learning *i.e.* the data-set used for the VAT-loss and for the ASR-loss where the same. For end-to-end speech recognition models, such as AED models, the actual word sequence is used as the label, hence VAT can be more successful as the labels are more well-defined.

6.3 Cosine Distance VAT for Speaker Verification

6.3.1 Speaker Classification and Speaker Verification

In speaker classification, there are S different speakers. For each speaker, there is some training data available. During the evaluation, a new utterance is presented and the task is to determine which of the S speakers is speaking. In speaker verification, the task is to determine whether the presented utterance belongs to the speaker that it is claimed to be from *i.e.* the prediction is binary. In this work, the slightly more narrow definition of speaker verification is used, where the system verifies whether a pair of utterances are from the same speaker or not.

6.3.2 Speaker embeddings

To determine whether the speakers of a pair of utterances match, speaker-discriminative embeddings (also called speaker embeddings) of the two utterances are compared. A speaker embedding is a fixed dimensional vector that is derived from an utterance and contains predominantly information that can be used to differentiate who is speaking.

Speaker embeddings derived through deep learning techniques have become the state-of-the-art for learning speaker representations (Cyrta et al., 2017; Snyder et al., 2017; Snyder et al., 2018) to be used for tasks such as speaker recognition, speaker verification or speaker diarisation (Díez et al., 2019; Li et al., 2021c; Snyder et al., 2017; Villalba et al., 2020). Previously i-vectors (Dehak et al., 2011; Sell and Garcia-Romero, 2014; Shum et al., 2011) based on factor analysis were widely used.

The neural networks used to generate speaker embeddings are typically trained on a speaker classification task, for which the input is the acoustic feature sequence of an utterance and the output is the speaker label of that utterance (Snyder et al., 2017; Variani et al., 2014). By taking the output of a layer of this neural network (often the penultimate layer) as an embedding, a fixed dimensional vector can be generated for any given input utterance. This vector is speaker-discriminative due to the training objective. It has been found that such speaker embeddings can be used to discriminate between speakers that are not present in the training data.

For training speaker embeddings, the angular softmax loss has become very popular (Fathullah et al., 2020; Huang et al., 2018; Luu et al., 2020) as well as for face verification (Deng et al., 2019; Liu et al., 2017).

An advantage of using the angular softmax loss is that the cosine similarity between two embeddings can be used as a score for the similarity of the speakers (Li et al., 2018; Luu et al., 2020). This work uses the angular softmax loss for training and cosine similarity for scoring. The difference between the standard softmax and the angular softmax is that the bias vector is set to zero and that the rows of the weight-matrix are constraint to have an L_2-norm of one.

6.3.3 Evaluation Metrics for Speaker Verification

The primary metric for evaluating speaker verification systems is the Equal Error Rate (EER). For speaker verification, the test-set consists of utterance pairs and the task is to determine whether or not they belong to the same speaker. Based on the speaker-embeddings a score is calculated (in this work the cosine similarity is used). A higher score should indicate that the two utterances belong to the same speaker and a low score should indicate that the two utterances belong to different speakers. A threshold ϕ is chosen to determine the cut-off for the binary classification *i.e.* if the score is above ϕ the two utterances are determined to come from the same speaker, if the score is below ϕ the two utterances are determined to come from different speakers.

Given a threshold ϕ, the empirical probability of false positives and the probability of false negatives can be calculated. The Equal Error Rate (EER) is a commonly used error metric that is obtained by sweeping thresholds until a value, ϕ_{EER}, is found such that the rate of false positives equals the rate of false negatives.

Speaker-discriminative acoustic embeddings should have two qualities: (i) intra-speaker compactness (ISC) *i.e.* how close to each other are the embeddings of a single speaker; and (ii) inter-speaker separability (ISS) *i.e.* how far apart are the embeddings of different speakers. To give further insight into the embeddings generated by the trained models, measures of these two qualities are proposed. It is assumed that the embeddings $\boldsymbol{e}$ are L_2 normalised. Both metrics are calculated on the unseen test-set that contains unseen speakers. First, all utterance embeddings belonging to the same speaker are collected and the normalised centroid, $\mathbf{c}_i$, is calculated (one per speaker).

$$\begin{aligned} \tilde{\mathbf{c}_i} &= \sum_{e \in \mathcal{E}_i} \boldsymbol{e} \\ \mathbf{c}_i &= \frac{\tilde{\mathbf{c}_i}}{\|\tilde{\mathbf{c}_i}\|} \end{aligned} \tag{6.15}$$

where i is the speaker, $\boldsymbol{e}$ is a speaker-discriminative embedding of an utterance (speaker embeddings), and $\mathcal{E}_i$ is the set of speaker embeddings belonging to speaker i. $\mathcal{E}_i$ is constructed by obtaining the embeddings of every utterance in the test-set that are from speaker i.

For the ISS metric, the average pairwise cosine-distance between the centroids of all speakers is found,

$$\text{ISS} = \frac{2}{(S-1)(S)} \sum_{i=1}^{S-1} \sum_{j=i+1}^{S} \text{cd}[\mathbf{c}_i, \mathbf{c}_j] \tag{6.16}$$

where S is the number of speakers and $\text{cd}[\cdot, \cdot]$ is the cosine distance *i.e.*

$$\text{cd}[\mathbf{a}, \mathbf{b}] = \frac{1}{2} - \frac{\mathbf{a}^T \mathbf{b}}{2\,\|\mathbf{a}\|\,\|\mathbf{b}\|} \tag{6.17}$$

For the ISC, all utterance embeddings belonging to the same speaker are collected, the average distance to the centroid of that speaker is found, and the average of those per-speaker scores is calculated.

$$\mathrm{ISC} = \frac{1}{S}\sum_{i=1}^{S}\frac{1}{\|\mathcal{E}_i\|}\sum_{e\in\mathcal{E}_i}\mathrm{cd}[\mathbf{c}_i, \boldsymbol{e}] \tag{6.18}$$

6.3.4 CD-VAT loss

Here, VAT will be formulated on the level of the embedding layer rather than the output (classification) layer (see Section 6.1 for details). The reason for this is that the unsupervised data set $\mathcal{D}_{ul}$ contains different speakers from the supervised data set $\mathcal{D}_l$. This section proposes cosine-distance virtual adversarial training (CD-VAT). The purpose of this proposed variant of VAT is to smooth the embedding generator in terms of the cosine distance. CD-VAT should be used together with an angular penalty loss (such as angular softmax (Liu et al., 2017)) in comparison to the standard cross-entropy loss. As mentioned before, when angular penalty losses are used during speaker classification training, the resulting embedding generator produces embeddings that are angularly discriminative *i.e.* the cosine distance between embeddings indicates if embeddings come from the same speakers. Therefore, here, the measure used for defining the smoothness matches the evaluation criteria of the speaker verification system. This means that no additional training is required for the speaker verification system given that the speaker embeddings can be directly used for scoring.

As in VAT, CD-VAT adds an additional cost $\mathcal{R}_{\text{CDVAT}}(\mathcal{D}_l, \mathcal{D}_{ul}, \theta)$ (the CD-VAT cost) to the supervised cost $\mathcal{C}_{\text{S}}(\mathcal{D}_l, \theta)$ with the interpolation constant α and is computed on both the labelled data set $\mathcal{D}_l$ (size L) and the unlabelled data set $\mathcal{D}_{ul}$ (size U). The combined cost $\mathcal{C}(\mathcal{D}_l, \mathcal{D}_{ul}, \theta)$ is then used to train the parameters θ and in turn the embedding generator $\mathbf{e}(\mathbf{x}, \theta)$.

$$\mathcal{C}(\mathcal{D}_l, \mathcal{D}_{ul}, \theta) = \mathcal{C}_{\text{S}}(\mathcal{D}_l, \theta) + \alpha\mathcal{R}_{\text{CDVAT}}(\mathcal{D}_l, \mathcal{D}_{ul}, \theta) \tag{6.19}$$

The CD-VAT loss, $\mathcal{R}_{\text{CDVAT}}$, is the sum of local, per data point, losses $\mathrm{LCS}(\mathbf{x}, \theta)$ that are computed for each input feature sequence $\mathbf{x} \in \mathcal{D}_l, \mathcal{D}_{ul}$.

$$\mathcal{R}_{\text{CDVAT}}(\mathcal{D}_l, \mathcal{D}_{ul}, \theta) = \frac{1}{L+U}\sum_{\mathbf{x}\in\mathcal{D}_l,\mathcal{D}_{ul}}\mathrm{LCS}(\mathbf{x}, \theta) \tag{6.20}$$

The local cosine smoothness, $\mathrm{LCS}(\mathbf{x}, \theta)$, is calculated in two steps. First, a perturbation ($\mathbf{r}_{\text{CDVAT}}$) to the input sequence $\mathbf{x}$ is found. This perturbation is chosen to be an *adversarial* perturbation that maximally changes the embedding ($\mathbf{e}(\mathbf{x}+\mathbf{r}_{\text{CDVAT}}, \theta)$) of the input feature sequence, $\mathbf{x}$, as measured by the cosine-distance (cd$[\cdot,\cdot]$; Equation (6.17)). ϵ is the maximum norm of the adversarial perturbation $\mathbf{r}_{\text{CDVAT}}$.

$$\mathbf{r}_{\text{CDVAT}} = \underset{\mathbf{r};\|\mathbf{r}\|\leq\epsilon}{\arg\max}\, \mathrm{cd}[\mathbf{e}(\mathbf{x},\theta)\,, \mathbf{e}(\mathbf{x}+\mathbf{r},\theta)] \tag{6.21}$$

Second, $\mathrm{LCS}(\mathbf{x},\theta)$ is then the cosine distance between the embedding of the (maximally) perturbed input sequence and the embedding of the original unperturbed input sequence.

$$\mathrm{LCS}(\mathbf{x},\theta) = \mathrm{cd}[\mathbf{e}(\mathbf{x},\theta')\,, \mathbf{e}(\mathbf{x}+\mathbf{r}_{\text{CDVAT}},\theta)] \tag{6.22}$$

θ' is the current setting for θ at a particular instant during optimisation *i.e.* it is treated as a constant. The distinction between θ and θ' is made because gradients of $\mathrm{LCS}(\mathbf{x},\theta)$ are only propagated back through the embedding generated with the input perturbation *i.e.* $\mathbf{e}(\mathbf{x}+\mathbf{r}_{\text{CDVAT}},\theta)$ and not $\mathbf{e}(\mathbf{x},\theta')$. $\mathrm{LCS}(\mathbf{x},\theta)$ indicates how "sensitive" the embedding of input $\mathbf{x}$ is to an adversarial perturbation.

6.3.5 Gradient of the cosine-distance

As in Section 6.1.1, the virtual adversarial perturbation, $\mathbf{r}_{\text{CDVAT}}$, can be found using the iterative procedure. Given that now the distance metric is calculated at the level of the embedding, the gradient calculations are slightly different.

For simplicity let $\mathbf{g}_{i+1} = \nabla_{\mathbf{r}}\mathrm{cd}(\mathbf{r},\mathbf{x},\theta)\,|_{\mathbf{r}=\zeta\mathbf{v}_i}$. To calculate $\mathbf{g}_{i+1}$ the following gradients are needed:

$$\mathbf{g}_{i+1} = \left.\frac{\partial \mathrm{cd}(\mathbf{r},\mathbf{x},\theta)}{\partial \mathbf{r}}\right|_{\mathbf{r}=\zeta\mathbf{v}_i} \tag{6.23}$$

$$= \left.\frac{\partial \mathbf{e}(\mathbf{x}+\mathbf{r},\theta)}{\partial \mathbf{r}}\frac{\partial \mathrm{cd}(\mathbf{r},\mathbf{x},\theta)}{\partial \mathbf{e}(\mathbf{x}+\mathbf{r},\theta)}\right|_{\mathbf{r}=\zeta\mathbf{v}_i} \tag{6.24}$$

The pre-multiplication with $\frac{\partial \mathbf{e}(\mathbf{x}+\mathbf{r},\theta)}{\partial \mathbf{r}}$ in Equation (6.24) is equivalent to the standard back-propagation algorithm, but the back-propagation starts from the embedding and not from the pre-softmax activations.

For calculating $\left.\frac{\partial \mathrm{cd}(\mathbf{r},\mathbf{x},\theta)}{\partial \mathbf{e}(\mathbf{x}+\mathbf{r},\theta)}\right|_{\mathbf{r}=\zeta\mathbf{v}_i}$, let $\mathbf{e}(\mathbf{x},\theta) = \mathbf{e}$ and $\mathbf{e}(\mathbf{x}+\mathbf{r},\theta) = \mathbf{e}_r$,

$$\frac{\partial \text{cd}(\mathbf{r}, \mathbf{x}, \theta)}{\partial \mathbf{e}_r} = \frac{\text{-}1}{2\left\|\mathbf{e}\right\|} \cdot \frac{\partial}{\partial \mathbf{e}_r}\left(\frac{\mathbf{e}^T\mathbf{e}_r}{\left\|\mathbf{e}_r\right\|}\right) \tag{6.25}$$

$$\begin{aligned} &= \frac{\text{-}1}{2\left\|\mathbf{e}\right\|} \cdot \left(\frac{1}{\left\|\mathbf{e}_r\right\|} \cdot \frac{\partial}{\partial \mathbf{e}_r}\left(\mathbf{e}^T\mathbf{e}_r\right)\right. \\ &\quad \left. + \frac{\partial}{\partial \mathbf{e}_r}\left(\frac{1}{\left\|\mathbf{e}_r\right\|}\right) \cdot \left(\mathbf{e}^T\mathbf{e}_r\right)\right) \end{aligned} \tag{6.26}$$

$$= \frac{\text{-}1}{2\left\|\mathbf{e}\right\|} \cdot \left(\frac{1}{\left\|\mathbf{e}_r\right\|} \cdot \mathbf{e} - \frac{\mathbf{e}_r}{\left\|\mathbf{e}_r\right\|^3} \cdot \left(\mathbf{e}^T\mathbf{e}_r\right)\right) \tag{6.27}$$

The gradient equation from Equation (6.27) is also used to calculate the gradients for the local-cosine smoothness.

6.3.6 Experimental Setup

Experiments were designed to evaluate the effect of CD-VAT on general speaker verification performance, while also more directly testing the effect of CD-VAT on intra-speaker compactness and inter-speaker separability. These are measured using the ISC and ISS metrics that were proposed in Section 6.3.3.

6.3.6.1 Data Sets

Two data sets, VoxCeleb1 (dev+test) (Nagrani et al., 2017) and VoxCeleb2 (dev+test) (Chung et al., 2018; Nagrani et al., 2019) were used to train the models and evaluate speaker verification performance. The VoxCeleb data sets consist of utterances that were obtained from YouTube videos and automatically labelled using a visual speaker recognition pipeline. The VoxCeleb2 (dev+test) data set, together with the dev portion of the VoxCeleb1 data set, was used for training. For evaluation, the test portion of VoxCeleb1 was used. The combined train set consists of more than 2750 hours of data from 7323 different speakers. The evaluation set consists of 4874 utterances from 40 speakers, for which the official speaker verification list of 37720 utterance pairs was used. More information about the data is contained in Table 6.2 and in Section A.1.

The system input features were 30-d Mel-scale Frequency Cepstral Coefficients (MFCCs). The MFCCs are extracted (using HTK (Young et al., 2015)) using 25 ms windows with 10 ms frame increments from 30 filterbank channels. These inputs were normalised at the utterance level for mean and globally for variance. No forms of data augmentation were used for these experiments.

Title	train	test
# Speakers	7323	40
# Videos	172299	677
# Utterances	1276888	4874
Avg. Utterance Length	7.85 sec	8.25 sec
Total Length	2784 h	11 h

Table 6.2 The train data is the combination of VoxCeleb2 (dev+test) and the development portion of VoxCeleb1. The test data is the test portion of VoxCeleb1 for which 37720 utterance pairs are the verification list.

6.3.6.2 Model Architecture

In the speaker embedding model, utterance-level speaker embeddings are created by averaging multiple L_2-normalised window-level embeddings. The input window to the window-level embedding generator is around 2 seconds (213 frames, [-106,+106]) long. The shift between windows is just under 100 frames (see details below). The embedding generator uses a TDNN (Peddinti et al., 2015) with a total input context of [-7,+7], which is shifted from {-99} to {+99} with shifts of 6 frames (resulting in the overall input window of [-106,+106]). The 34 output vectors of the TDNN are combined using the self-attentive layer proposed in (Sun et al., 2019). This is followed by a linear projection down to the embedding size, which is then the window-level embedding. The TDNN structure resembles the one used in the x-vector models (Snyder et al., 2018) (*i.e.* TDNN-layers with the following input contexts: [-2,+2], followed by {-2,0,+2}, followed by {-3,0,+3}, followed by {0}). The first three TDNN-layers have a size of 512, the third a size of 256 and the embedding size is 32.

An utterance of length T fits $N = \lceil \frac{T-213}{100} \rceil$ full windows (at shifts of 100 frames) plus another window if padding is used (*e.g.* replication padding to $213+N*100$ frames). To avoid padding, shifts of $\frac{T-213}{N}$ are used (*i.e.* slightly under 100 frames). The resulting indices are rounded to the nearest integer. For utterances shorter than 213 frames, the window is aligned to the centre of the utterance and replication padding is used.

6.3.6.3 Training

CD-VAT was implemented in HTK (Young et al., 2015) with which all models were trained in conjunction with PyHTK (Zhang et al., 2019). For training, the window-level embedding is classified into the different speakers. The training objective for supervised training is the angular softmax (Liu et al., 2017) with $m = 1$. The embedding generator was optimised using Stochastic Gradient Descend (SGD) with momentum, and weight-decay was used for regularisation. The learning rate scheduler was NewBob. The batch size used for the supervised loss, $\mathcal{C}_{\mathrm{S}}(\mathcal{D}_l, \theta)$, was 200 except for the model trained on the entire data set for which a batch size of 400 was used. The batch size used for the CD-VAT loss, $\mathcal{R}_{\text{CDVAT}}(\mathcal{D}_l, \mathcal{D}_{ul}, \theta)$, was 800 *i.e.* four times higher. The interpolation coefficient α was set to 0.4 and the norm of the adversarial perturbation ϵ was set to 13. The model was trained directly on the combined cost, $\mathcal{C}\,(\mathcal{D}_l, \mathcal{D}_{ul}, \theta)$, *i.e.* not pre-trained on the purely supervised loss.

For the experiments, the data was partitioned into labelled and unlabelled data in two different ways. For the first one, 220k utterances were labelled, and for the second one, 440k utterances were labelled. The remaining 1057k and 837k utterances, respectively, form the unlabelled data sets. The 220k and 440k utterances were chosen from the top of the utterance list that was sorted by the official utterance name. This means that the labelled and unlabelled data have at most one speaker in common. Of the labelled data set, 20k and 40k utterances, respectively, form the validation set.

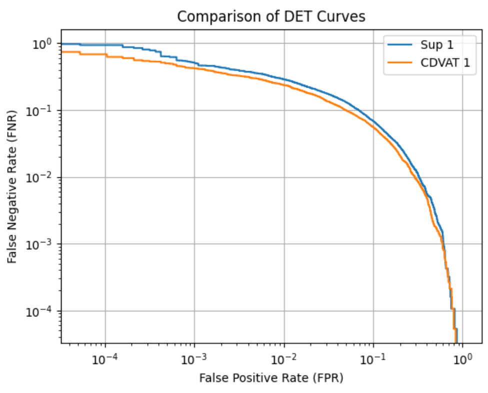
6.3.7 Experimental Results

System	Utterances in $\mathcal{D}_l$	Speakers in $\mathcal{D}_l$	Utterances in $\mathcal{D}_{ul}$	EER
Sup 1	200k	1249	-	8.32%
CDVAT 1			1057k	7.40%
Sup 2	400k	2504	-	6.85%
CDVAT 2			837k	6.46%
Sup 3	1277k	7323	-	5.52%

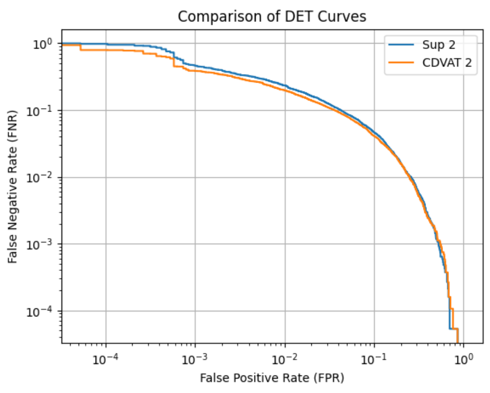
Table 6.3 Evaluation of CD-VAT on the VoxCeleb dataset. The evaluation criterion Equal Error Rate (EER) is explained in Section 6.3.3. For EER, a lower value is better. $\mathcal{D}_l$ is the labelled data set and $\mathcal{D}_{ul}$ is the unlabelled data set.

Tables 6.3 and 6.4 show the results of applying CD-VAT on the VoxCeleb data set. The supervised baseline model trained on 200k utterances of labelled data achieved an EER of 8.32%. The use of CD-VAT reduced the EER by 11.1% relative down to an EER of 7.40%. This represents an EER recovery of 32.5% *i.e.* 32.5% of the reduction in EER that would result from purely supervised training if the speaker labels for the unlabelled part of the training data were available (such a model has an EER of 5.52%). This error rate recovery is similar to those seen in other areas of machine learning. For instance, Manohar et al. (2015) present a word error rate recovery of 37% for semi-supervised learning (minimum entropy training) of a HMM-DNN speech recogniser without additional language modelling data. At the same time, semi-supervised speaker embeddings present additional challenges as for larger numbers of speakers, class overlap can exist. The information content of unlabelled examples decreases as classes overlap (Castelli and Cover, 1996; O'Neill, 1978).

The supervised baseline model trained on 400k utterances of labelled data achieves an EER of 6.85%. The use of CD-VAT reduces the EER by 5.7% relative down to an EER of 6.46%. This represents an EER recovery of 29.3%.



(a) 200k utts in $\mathcal{D}_l$ experiment



(b) 400k utts in $\mathcal{D}_l$ experiment

Figure 6.1 Comparisons of Detection Error Tradeoff (DET) curves with and without CD-VAT.

Furthermore, Figure 6.1 shows the DET curves for the experiments that use 200k utterances and 400k utterances as the supervised data set. The curves show that VAT is more useful when the ration of unsupervised to supervised data set is larger. The curves also show that VAT is more useful for thresholds that favour lower false positive rates (higher threshold) rather than favouring lower false negative rates. This is not surprising, given that VAT, as shown by the lower ISC values, produces more compact groups of speakers and can thus still produce True positives even for very high thresholds.

Furthermore, the ISC, which is very closely related to the CD-VAT smoothing loss, is reduced for the 200k model from 0.13 down to 0.09 and for the 400k model from 0.14 down to 0.09. These are relative reductions of 31% and 36%, respectively. However, at the same time, the ISS was also slightly reduced. For the 200k model, the ISS reduces from 0.38 to 0.36 and for the 400k model, from 0.40 to 0.38. This shows one disadvantage of CD-VAT, which is that it also brings the embeddings of all utterances closer together. To put these values into perspective, the threshold of the cosine-scoring used to obtain the EER is between 0.42 and 0.48 for these systems *i.e.* the threshold for cosine-scoring is roughly the same as the average cosine-distance between the centroids of the test-set speakers.

System	Utterances in $\mathcal{D}_l$	Utterances in $\mathcal{D}_{ul}$	ISC	ISS
Sup 1	220k	-	0.13	0.38
CDVAT 1		1057k	0.09	0.36
Sup 2	440k	-	0.14	0.40
CDVAT 2		837k	0.09	0.38
Sup 3	1277k	-	0.14	0.46

Table 6.4 Evaluation of CD-VAT on the VoxCeleb dataset. The evaluation criteria intra-speaker compactness (ISC) and inter-speaker separability (ISS) are explained in Section 6.3.3. For ISC a lower value is better. For ISS a higher value is better.

6.4 Summary

This chapter focused on applying consistency regularisation to speech recognition and to the training of speaker embeddings.

In Section 6.2, it was shown that VAT is not easily usable for HMM-DNN models for which the supervised loss is CE training on the frame-level due to inherent label noise. This was shown by demonstrating that for a strong weight of the VAT-loss, label-smoothing improved the learning.

In Section 6.3, the novel algorithm CD-VAT was developed. It allows for semi-supervised training of speaker-discriminative acoustic embeddings without the requirement that the set of speakers is the same for the labelled and the unlabelled data. It is shown that CD-VAT improves speaker verification performance on the VoxCeleb data set over a purely supervised baseline. For an experimental setup where 16% or 200k of the data utterances are treated as labelled and the other 84% are treated as unlabelled, the proposed method recovers 32.5% of the EER improvement that is obtained when all speaker labels are available for the unlabelled data. At the same time, however, the computational cost of CD-VAT is twice as high (per data point) as supervised training, and two new hyper-parameters that need to be tuned were introduced.

Furthermore, Section 6.3.3 introduced two new metrics for analysis of a speaker embedding model: (i) a metric for the intra-speaker compactness (ISC); and (ii) a metric for the inter-speaker separability (ISS). These metrics are highly general, not limited to semi-supervised learning and can be used in other scenarios. For other training objectives, the cosine-distances in Equations (6.16) and (6.18) could be replaced with the Euclidean distance.

At the time of its original publication, CD-VAT was the first algorithm for semi-supervised learning for speaker verification. Since then, self-supervised learning algorithms have been getting increasing amounts of attention. They have now been used for semi-supervised learning for speaker verification with excellent results (Chen et al., 2022; Yang et al., 2021). During the fine-tuning stage that always follows the self-supervised representation learning, CD-VAT can still be used. This would be a promising future direction to explore.

Chapter 7

Bayesian Active Learning for ASR

This chapter introduces an approach to active learning for Automatic Speech Recognition (ASR) based on Bayesian principles. The method selects utterances based on how much they could reduce the uncertainty about the parameters and not based on the model's uncertainty about the transcription. This is important because uncertainty about the transcription can arise from simply having noisy input data. A large number of possible parameter settings may be able to model a given data set, in which case there is uncertainty about the model's parameters. In Bayesian active learning, the objective is to choose data that will reduce this uncertainty. This means that input utterances $\mathbf{X}_i$ should be selected based on the mutual information between the prediction for the utterance and the model parameters $\boldsymbol{\theta}$, *i.e.* $\mathcal{I}[\theta, \mathrm{w}_i|\mathcal{D}_l, \mathrm{X}_i]$.

In this chapter, the first section covers the background for using the mutual information $\mathcal{I}[\theta, \mathrm{w}_i|\mathcal{D}_l, \mathrm{X}_i]$ as an acquisition function for active learning as proposed by Houlsby et al. (2011). Then methods are derived and proposed for how to apply this acquisition function for sequence-to-sequence models as used in ASR. In particular, NBestBALD is proposed as a method to implement Bayesian active learning for ASR. Given issues with NBestBALD it is combined with a representativeness approach. Experimental results using the SwitchBoard (SWB) dataset show that NBestBALD outperforms the other examined methods.

7.1 Background

This section describes the background regarding Bayesian active learning and the acquisition function that will be used.

7.1.1 Information gain utility function

In Bayesian active learning, the goal is to choose new data points that are the most useful for learning about the model parameters θ. Learning about the model parameters can be quantified using the Shannon entropy (see Equation (2.55)) over the parameters.

In the Bayesian framework, model parameters are seen as random variables, and learning constitutes building a posterior distribution $p(\theta \mid \mathcal{D}_l)$ over the parameters, θ, given the labelled data set $\mathcal{D}_l = \{(\mathbf{x}_i; \hat{\mathbf{y}}_i)\}_{i=1}^{L}$. Ideally, this probability distribution over the model parameters is tractable, such that it can be used to calculate the entropy $\mathbb{H}[\theta \mid \mathcal{D}_l]$, which expresses the uncertainty of the posterior about the parameters.

The amount of information that is gained by including a single new data pair $(\mathbf{x}_i, \hat{y}_i)$ is then given by the difference between the entropy before and after including the new data pair $(\mathbf{x}_i, \hat{y}_i)$. For active learning, this principle can be used by designing an acquisition function that estimates this information gain. This estimation is given by:

$$f(\mathbf{x}_i) = \mathbb{H}[\theta \mid \mathcal{D}_l] - \mathbb{E}_{P(y_i \mid \mathbf{x}_i, \mathcal{D}_l)}[\mathbb{H}[\theta \mid \mathcal{D}_l, \mathbf{x}_i, y_i]] \tag{7.1}$$

The expectation over the label y_i has to be taken because the true label is unknown at the time at which the acquisition function is evaluated. Another intuitive approach would be to maximise the Kullback-Leibler divergence (KL-divergence) between $p(\theta \mid \mathcal{D}_l)$ and $p(\theta \mid \mathcal{D}_l, \mathbf{x}_i, \hat{y}_i)$. It turns out that data points that maximise this KL-divergence also maximise to Equation (7.1).

7.1.2 Reformulation of the information gain

This section presents a reformulation of Equation (7.1) as presented by Houlsby et al. (2011). This reformulation stems from the problem that the predictive distribution, $P(y \mid \mathbf{x}_i, \mathcal{D}_l)$, and the posterior, $p(\theta \mid \mathcal{D}_l)$, are not usually in a tractable form such that it can be used to evaluate the expectation of the entropy. The information gain given in Equation (7.1) is equivalent to the mutual information between the parameters, θ, and the unobserved output, y_i, conditioned upon the input data point, $\mathbf{x}_i$, and the supervised data set, $\mathcal{D}_l$, *i.e.* the information gain is $\mathcal{I}[\theta; y_i \mid \mathcal{D}_l, \mathbf{x}_i]$.

Because the mutual information is symmetric, Equation (7.1) can be rewritten as:

$$
\begin{aligned}
f(\mathbf{x}_i) &= \mathbb{H}\left[\theta \mid \mathcal{D}_l\right] - \mathbb{E}_{P(y_i \mid \mathbf{x}_i, \mathcal{D}_l)}\left[\mathbb{H}\left[\theta \mid \mathcal{D}_l, \mathbf{x}_i, y_i\right]\right] && (7.1)\\
&= \mathcal{I}\left[\theta; y_i \mid \mathcal{D}_l, \mathbf{x}_i\right] && (7.2)\\
&= \mathbb{H}\left[y_i \mid \mathbf{x}_i, \mathcal{D}_l\right] - \mathbb{E}_{\theta \mid \mathcal{D}_l}\left[\mathbb{H}[y_i \mid \mathbf{x}_i, \theta]\right] && (7.3)
\end{aligned}
$$

The first term in Equation (7.3), is the entropy of the trained model over the output y_i for input $\mathbf{x}_i$. This is the uncertainty that the model has over the label of the data point $\mathbf{x}_i$. The second term is the expectation of the conditional entropy. It represents the average uncertainty over the label of the data point $\mathbf{x}_i$ given a specific parameter setting. Therefore, the acquisition function of Equation (7.3) will choose input data points for which the model has a high output entropy but a small expected conditional entropy. The entropy of the predictive distribution $\mathbb{H}\left[y_i \mid \mathbf{x}_i, \mathcal{D}_l\right]$ stems from a combination of the parameter uncertainty and the inherent noise. The expected conditional entropy, $\mathbb{E}_{p(\theta \mid \mathcal{D}_l)}\left[\mathbb{H}\left[y_i \mid \mathbf{x}_i, \theta\right]\right]$, mostly captures the inherent noise of the task. It measures the entropy when the parameters have been found. If the entropy is still high given that the parameters are found this means that the task is noisy.

Therefore, the acquisition function of Equation (7.3) should select data points for which the uncertainty stems from the uncertainty over the parameters and not from the uncertainty due to the inherent noise. From another viewpoint, Equation (7.3) has a high value if the parameter settings of θ that are probable under the posterior are individually confident (low expected conditional entropy) but highly diverse such that, as a whole, the model is still uncertain (high entropy of the predictive distribution). This means that the parameter settings disagree about the output y_i. Therefore, Houlsby et al. (2011) termed this acquisition function Bayesian Active Learning by Disagreement (BALD). Based on Equation (7.3) BALD is dependent on the data and the model, but not the training algorithm used to infer the model's parameters.

7.1.3 Monte-Carlo Estimation

The BALD acquisition function given in Equation (7.3) requires a posterior distribution over the parameters, $p(\theta \mid \mathcal{D}_l)$, in order to compute the expected conditional entropy, $\mathbb{E}_{p(\theta \mid \mathcal{D}_l)}\left[\mathbb{H}\left[y_i \mid \mathbf{x}_i, \theta\right]\right]$. Even if the posterior is tractable, this entropy is still not often tractable. A Monte-Carlo approximation can be used to approximate Equation (7.3) because given a specific parameter setting, θ, the entropy $\mathbb{H}\left[y_i \mid \mathbf{x}_i, \theta\right]$ is easy to calculate as it is usually a simple categorical distribution. Furthermore, it is not required to have access to the entire distribution $p(\theta \mid \mathcal{D}_l)$; instead, it is only necessary to have access to samples of this distribution.

The Monte-Carlo approximation of the expectation of the entropy is given by:

$$\mathbb{E}_{p(\theta|\mathcal{D}_l)}\left[\mathbb{H}[y_i \mid \mathbf{x}_i, \theta]\right] \approx \frac{1}{J}\sum_{n=1}^{J} \mathbb{H}[y_i \mid \mathbf{x}_i, \theta_n], \theta_n \sim q(\theta) \tag{7.4}$$

where J is the number of samples that is used and $q(\theta)$ is the variational distribution that is used to approximate $p(\theta \mid \mathcal{D}_l)$. The larger J is the lower the variance of this Monte-Carlo estimate is going to be.

As discussed in Section 2.4.3, it was derived by Gal and Ghahramani (2016) that in a model that has been trained with dropout, using a dropout mask can be seen as sampling from $q(\theta)$, which in turn can be interpreted as a Bayesian approximation to Gaussian processes (Rasmussen and Williams, 2006).

Similar to Equation (7.4), the entropy of the predictive distribution $\mathbb{H}\left[y_i \mid \mathbf{x}_i, \mathcal{D}_l\right]$ can be approximated as:

$$\mathbb{H}\left[y_i \mid \mathbf{x}_i, \mathcal{D}_l\right] \approx \mathbb{H}_{\frac{1}{J}\sum_{n=1}^{J} P(y_i|\mathbf{x}_i,\theta_n)}\left[y_i \mid \mathbf{x}_i, \theta_n\right], \theta_n \sim q(\theta) \tag{7.5}$$

Therefore, the Monte-Carlo estimate of the BALD acquisition function becomes:

$$f(\mathbf{x}_i) = \mathbb{H}_{\frac{1}{J}\sum_{n=1}^{J}(P(y_i|\mathbf{x}_i,\theta_n))}\left[y_i \mid \mathbf{x}_i, \theta_n\right] - \frac{1}{J}\sum_{n=1}^{J} \mathbb{H}\left[y_i \mid \mathbf{x}_i, \theta_n\right], \theta_n \sim q(\theta) \tag{7.6}$$

Instead of using dropout, an ensemble can also be used (Lakshminarayanan et al., 2017), where each model is viewed as a sample from the posterior. This requires to train multiple models for each iteration of training. Given the efficiency of the dropout-based approach, the ensemble approach was not used in this thesis.

7.2 BALD for ASR

This section presents how BALD can be used in ASR models. It presents a formulation for the sequence-to-sequence distributions that are used in ASR, a method to calculate the entropy needed for the acquisition function and how it can be combined with a representativeness measure that follows the information density framework (see Section 5.4.4).

7.2.1 Acquisition function for sequence-to-sequence learning

In Section 7.1, the BALD acquisition function was derived for simple 1-1 mapping problems with a single input vector $\mathbf{x}$ and a single output label y. In ASR the input is presented as a sequence of vectors $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_T]$ and the output is the word-sequence $\mathbf{w} = [w_1, ..., w_M]$.

The available data consists of both L labelled instances $\mathcal{D}_l = \{(\mathbf{X}_i; \hat{\mathbf{w}})\}_{i=1}^{L}$ and U unlabelled instances $\mathcal{D}_{ul} = \{\mathbf{X}_i\}_{l=L+1}^{L+U}$, where often $U \gg L$. This leads to the acquisition function:

$$f(\mathbf{X}) = \mathbb{H}\left[\mathbf{w}|\mathbf{X}, \mathcal{D}_l\right] - \mathbb{E}_{p(\theta|\mathcal{D}_l)}\left[\mathbb{H}[\mathbf{w}|\mathbf{X}, \theta]\right] \tag{7.7}$$

and with Monte-Carlo dropout:

$$f(\mathbf{X}) = \mathbb{H}_{\frac{1}{J}\sum_{n=1}^{J} P(\mathbf{w}|\mathbf{X},\theta_n)}\left[y_i \mid \mathbf{x}_i, \theta_n\right] - \frac{1}{J}\sum_{n=1}^{J}\mathbb{H}[\mathbf{w}|\mathbf{X}, \theta_n] \tag{7.8}$$

For the first term, the output distributions after applying multiple dropout masks have to be averaged. For the second term, the entropies of the distributions of multiple dropout masks are averaged.

7.2.2 Entropy calculation

The entropy of a sequence-to-sequence task, $\mathbb{H}[\mathbf{w}|\mathbf{X}, \theta_n]$, is not well defined because the model would have to define a distribution over an infinite set of variable-length output sequences. This thesis approximates the entropy using the N-Best list.

$$\mathbb{H}[\mathbf{w}|\mathbf{X}, \theta_n] = \sum_{\mathbf{w}\in\mathcal{W}} P(\mathbf{w}|\mathbf{X}, \theta_n)\log P(\mathbf{w}|\mathbf{X}, \theta_n) \tag{7.9}$$

$$\approx \sum_{i=1}^{N} g\left(\mathbf{w}_i|\mathbf{X}\right)\log g\left(\mathbf{w}_i|\mathbf{X}\right) \tag{7.10}$$

where $\mathbf{w}_i$ is one of the N-Best hypotheses and $g\left(\mathbf{w}_i|\mathbf{X}\right)$ is the normalised probability of the hypotheses such that the probabilities of the N-Best hypotheses add up to 1. Here, care has to be taken when normalising the probabilities in order to ensure numerical stability. This is done by first subtracting the maximum un-normalised log-probability from each of the un-normalised log-probabilities. For the entropy calculation, the acoustic model log-likelihood values are multiplied by the inverse of the Language Model (LM) scale factor as done in MMIE-Training (Woodland and Povey, 2002) or in lattice-based confidence estimation. This smoothes the distribution without affecting the 1-best hypothesis.

In initial experiments, the Lattice-Entropy was used instead. The entropy of a lattice can be calculated using the results of the forward-backward algorithm. The fact that the different lattices are pruned in different ways lead to the decision to use NBest-Lists instead.

7.2.3 Combination with representativeness measure

As discussed in Section 5.4.4, combining an informativeness score (such as BALD) with a representativeness score as done in the information density framework (Settles and Craven, 2008) (see Section 5.4.4) can reduce querying outliers,

$$f(\mathbf{X}_i) = f'(\mathbf{X}_i) \cdot \left(\frac{1}{U} \sum_{j=L+1}^{U+L} \mathtt{sim}(\mathbf{X}_j, \mathbf{X}_i) \right)^{\beta} \tag{5.10}$$

where, $f'(\cdot)$ can be any of the previously defined acquisition functions, $\mathtt{sim}(\cdot,\cdot)$ is any similarity measure between two samples and β is a coefficient that weights the two criteria.

Equation (5.10) requires a similarity measure. This chapter uses the dot-products of term frequency inverse document frequency (TF-IDF) vectors as the similarity measure *i.e.*

$$\text{sim}(\mathbf{X}_i, \mathbf{X}) = \text{cd}(\mathbf{m}(\mathbf{X}_i), \mathbf{m}(\mathbf{X})) \tag{7.11}$$

where $\text{cd}(\cdot,\cdot)$ is the cosine-similarity *i.e.* the normalised dot-product of the two TF-IDF vectors. These vectors have a length of K and are calculated as the multiplication of the term-frequency and the inverse document frequency of individual features λ_k that can occur in an utterance.

$$m_k(\mathbf{X}_i) = \text{tf}(\lambda_k, \mathbf{X}_i) \cdot \text{idf}(\lambda_k) \tag{7.12}$$

$$\text{tf}(\lambda_k, \mathbf{X}_i) = \frac{\text{counts of } \lambda_k \text{ in } \mathbf{X}_i}{\text{total number of features in } \mathbf{X}_i} \tag{7.13}$$

$$\text{idf}(\lambda_k) = \log\left(\frac{U}{1 + (\text{count of utts in } \mathcal{D}_{ul} \text{ containing } \lambda_k)} \right) \tag{7.14}$$

where $m_k(\mathbf{X}_i)$ is the k^{th} element of the TF-IDF vector $\mathbf{m}(\mathbf{X}_i)$, λ_k is a phone N-gram and U is the number of utterances in $\mathcal{D}_{ul}$. A phone N-gram λ_k is an N-gram within the phone-aligned 1-best hypotheses of the utterances. The set of phone N-grams is determined from phone-aligned training transcripts. The "total number of features in $\mathbf{X}_i$" is the number of unique N-grams in $\mathbf{X}_i$ that are part of the chosen N-gram set. The "count of utts in $\mathcal{D}_{ul}$ containing λ_k" is the number of utterances in the unlabelled data set in which the phone N-gram λ_k occurs.

The set of phone N-grams $\{\lambda_k\}_{k=1}^{K}$ is chosen based on uni-gram to 4-gram counts within the labelled training set. All of the N-grams that have a count higher than 10 in the labelled training set were chosen. When aligning the labelled training set, the reference transcript was used. When aligning the unlabelled pool of utterances, the recognised transcript from tri-gram Viterbi decoding was used. The phone-error rate of Viterbi decoding is lower than the phone-error rate of Confusion network decoding (which is used for decoding in the experiments), mainly due to an increase in deletions. Equation (7.11) can be efficiently implemented using sparse vectors. Using the TF-IDF vectors based on phone N-grams for similarity measurement in active learning for speech recognition has previously been used by Itoh et al. (2012). In this chapter, the interpolation coefficient β was set by matching the dynamic ranges of the TF-IDF scores and the informativeness scores.

7.3 Experimental Setup

This section presents the experimental setup that was used for the active learning experiments in this chapter. The majority of the experimental setup used for the experiments in Chapter 8 are also identical to this setup.

7.3.1 Data

The experiments were conducted using the Switchboard corpus (Switchboard-1 Release 2), a widely used multi-speaker corpus of conversational speech, for training. The Switchboard corpus consists of around 311 hours of data. Hub5’00 was used as the evaluation set, which is composed of two subsets: SWB and CallHome (CH). The SWB part of Hub5’00 consists of 20 conversations with 40 unique speakers, of which 36 appear in the training set. Given the overlap of speakers between the training set and the test set, which might cause biased results, these speakers were removed from the training set which leaves 305 hours of data. It is especially true for active learning experiments that choosing utterances for transcription that happen to be by a test speaker will result in evaluation results that are good but clearly biased. This leaves 484 training speakers. There is no overlap in the speakers of the training data and the CH portion of Hub5’00. More detail about the data sets and the exact difference between the data sets with and without the speaker overlap is given in Section A.3.

7.3.1.1 Initial set

To construct the initial supervised data set, a 20 hour subset of the data was selected from the 305 hours. This initial size of 20 hours was chosen such that three rounds of active data selection could be performed where the amount of training data doubles each time. Doubling the data allows us to have statistically significant differences in the model trained on the different data sets. Using less than 20 hours of data might lead to unrealistic results given that 20 hours is already a small training set.

In a realistic active learning scenario, such an initial small data set will likely only include a small number of speakers and the larger pool of unlabelled utterances will contain a larger number of speakers. To simulate this, the first 20 hours of data is not selected by randomly selecting utterances but instead by randomly sampling a speaker (without replacement) and then randomly selecting three sides of that speaker. A side refers to all the utterances of one speaker from one conversation. If fewer than three sides are available for that speaker, fewer sides are chosen *i.e.* one or two. This results in a data set containing 323 sides from 117 different speakers.

7.3.1.2 Selected sets

In order to thoroughly examine the effectiveness of the data selection methods, three iterations of batch acquisition are run for each method. The first batch has a size of 20 hours, the second has a size of 40 hours, and the third has a size 80 hours. These batches consist of individual utterances selected by the different acquisition methods that will be examined. Hence, the subsequent data set sizes are 40 hours, 80 hours and 160 hours *i.e.* the data doubles in each iteration.

7.3.2 Acoustic Model Training

The acoustic models that are used in the experiments are HMM-DNN hybrid ASR systems built using HTK (Young et al., 2015) together with (Zhang et al., 2019). First, an HMM-GMM triphone model is trained in order to generate state-level alignments for the frame-level Cross Entropy (CE) training of the hybrid systems. A 7-Layer FC-DNN is then trained on these alignments, used for re-alignment and then re-trained again on the new alignments. The 7-Layer FC-DNN takes the nine frames [-4,+4] (40d FBANK features plus Δ) as input. For all the 20 hours and 40 hour systems, the number of clustered tri-phone states is roughly 3000, and the width of the neural network (*i.e.* the size of the different layers) is exactly 1000. For the 80 hour and 160 hour systems, the number of clustered tri-phone states increases to roughly 6000, and the width of the neural network increases to exactly 2000. Weight-decay (see Section 2.3.1) and dropout (Srivastava et al., 2014) (see Section 2.3.2) are used for regularisation. All steps in the training pipeline (starting with a mono-phone HMM-GMM system) were run for every selected data set.

7.3.3 Language Models

For each selected data set for the various acquisition functions, a Language Model (LM) was trained on only the language modelling data. This language model was used for decoding the training set during the active learning procedure and also for evaluation. To describe the improvements to the acoustic model alone, the decoding results using another, much larger language model are also reported. This language model was trained on the training transcripts of both Switchboard and Fisher. The individual language models are called *specific LM*, and the large language model is called *SWB+Fisher LM* in the results section. For evaluation, confusion-network-decoding of a lattice rescored by a tri-gram language model was used for all results. The initial lattices were generated using a bi-gram language model. The perplexities of the tri-gram language models will be presented in Table 7.3.

7.3.4 Other examined active learning methods

The proposed NBest-BALD method of active selection of ASR data is compared to Random selection (at the utterance level) and two other active learning methods. The first uses Confidence scores. These were obtained via confusion network decoding (see Section 3.7). The average per-word confidence was used in the data selection process. The second uses the entropy of an NBest-List as the informativeness measure. For this NBest-Ent selection approach, the same representativeness encouraging approach from Section 7.2.3 as used for NBest-BALD was used here. For the Random selection for each selection three runs are done. The table reports the median of the runs and the maximum deviation from the median of the other two runs.

7.3.5 Decoding for Active Learning

For active learning using Confidence scores, NBest-Entropy and NBest-BALD, the unlabelled pool of utterances has to be decoded first. The confidence scores were obtained from confusion networks derived from decoding lattices that were rescored using a tri-gram language model. The NBest-List for the NBest-Entropy calculation was derived from a decoding lattice that was rescored using a tri-gram language model.

For NBest-BALD, one NBest-List for each entropy of Equation (7.8) is needed. To obtain the first NBest-List (for the first term in Equation (7.8)), multiple dropout masks were applied during decoding, and (as an approximation) the post-softmax activations were averaged. This is an approximation as ideally the sequence-level distributions should be averaged. Still, this approximation preserves the characteristic of the first term that for dropout masks with highly certain but very different outputs the value of the first term should be high. To get the second set of NBest-Lists (for the second term in Equation (7.8)), the decoding lattice used for the first term was rescored using acoustic model re-scoring with different dropout masks. Then, these lattices were used to create NBest-Lists and used to calculate the entropies.

For the TF-IDF-based similarity scores, the transcriptions generated from Viterbi decoding of a decoding lattice that was rescored using a tri-gram language model were used. The transcripts from the confusion-network decoding were not used. This is because confusion-network decoding results in a higher number of deletions. This, in turn, yields a higher phone error rate for the transcription, which is important given that phone N-grams are used in the TF-IDF vectors (see Section 7.2.3).

7.4 Experimental Results

The results from these experiments are shown in Tables 7.1 and 7.2. Table 7.1 contains the decoding results for the SWB and CH subsets of Hub5'00 where the language model used for decoding was trained on the transcripts of the full 311 hour Switchboard corpus and the much larger Fisher corpus. Here, the 20 hour baseline system has Word Error Rates (WERs) of 30.3% for SWB and 46.7% for CH. Here, the topline system that was trained all the data has WERs of 16.7% for SWB and 31.1% for CH. For the results in Table 7.2, only the data that was selected by the active learning algorithms were used to train the language model. Here, the 20 hour baseline system has WERs of 37.3% for SWB and 52.0% for CH. Here, the topline system that was trained all the data has WERs of 21.6% for SWB and 36.0% for CH. This difference in WER stems from the fact that the large SWB+Fisher language model is much better, it has a perplexity of 78.6 and the specific 20 hour language model has a perplexity of 129.8. The perplexities were calculated on the entire Hub5'00 evaluation set. Partial words were included in the calculation, though this did not make a meaningful difference in perplexity. The perplexities of the language models that were used - both for the specific language models and the large SWB+Fisher language model - are presented in Table 7.3.

7.4.1 40 hour results

For the 40 hour scenario, when using the large SWB+Fisher language model NBest-BALD achieves absolute WER reductions of 7.0% and 7.2% (on SWB and CH) over the 20 hour system (30.3% $\rightarrow$ 23.3% and 46.7% $\rightarrow$ 39.5%). These NBest-BALD absolute WER reductions are 17% and 9% higher relative to the Random selection baseline as well as 4% and 18% higher than for the Confidence score approach. Given that the experiments that use the large SWB+Fisher language model should examine the improvements to the acoustic model, this shows that for this operating point NBest-BALD improved the acoustic model more than the baseline approaches.

	SWB+Fisher LM			
	20 h	40 h	80 h	160 h
Random	30.3 / 46.7	24.3±0.4/ 40.1±0.6	19.9±0.4 / 35.1±0.2	17.8±0.4 / 32.9±0.4
Conf Score	-	23.6 / 40.6	19.5 / 34.9	17.5 / 32.4
NBest-Ent	-	23.6 / 39.7	19.3 / 34.7	17.3 / 32.3
NBest-BALD	-	23.3 / 39.5	19.2 / 34.2	17.1 / 32.0

Table 7.1 Active Learning results on the Hub5'00 set (SWB/CH). HMM-DNN acoustic models, tri-gram language model with confusion network decoding. The language model was trained using data from both the 311h Switchboard data set and the Fisher data set. For the random selection baseline the median of three runs and the maximum deviation of the other runs is shown.

	Specific LM			
	20 h	40 h	80 h	160 h
Random	37.3 / 52.0	31.4±0.2 / 47.4±0.4	27.1±0.4 / 42.3±0.2	24.0±0.3 / 39.2±0.2
Conf Score	-	31.6 / 47.2	26.7 / 42.3	23.4 / 38.9
NBest-Ent	-	31.6 / 47.2	26.7 / 41.9	23.2 / 38.5
NBest-BALD	-	31.7 / 47.1	25.7 / 41.3	23.0 / **38.2**

Table 7.2 Active Learning results on the Hub5'00 evaluation set (SWB/CH). HMM-DNN acoustic models, tri-gram language model with confusion network decoding. The language models are trained using the selected data only (specific LM). For the random selection baseline the median of three runs and the maximum deviation of the other runs is shown.

For the 40 hour scenario, using the specific language model NBest-BALD achieves absolute WER reductions of 5.6% and 4.9% (on SWB and CH) over then 20 hour system (37.3% $\rightarrow$ 31.7% and 52.0% $\rightarrow$ 47.1%). On the SWB subset these absolute WER reductions are 5% and 2% lower than for the Random baseline and the Confidence score approach, respectively. On the CH subset these absolute WER reductions are 7% and 2% higher than for the Random baseline and the Confidence score approach, respectively. Overall, for this operating point the active learning procedures did not consistently outperform the Random baseline. In fact, there was no statistically significant difference observed among all the different approaches. The perplexities in Table 7.3 give an indication towards the reason for this. The specific language model of the Random selection baseline has the lowest perplexity (which means it's better) on the evaluation sets. This could be because all active learning methods initially select a certain number of outliers. Many of these outliers are short utterances which help less with language

	Language models			
	20 h	40 h	80 h	160 h
Random	129.8	115.3±0.3	105.2±0.1	97.1±0.0
Conf Score	-	115.2	105.5	96.0
NBest-Ent	-	115.5	105.7	96.3
NBest-BALD	-	116.1	104.8	95.5
Full Train Set	90.6			
SWB+FSH	78.6			

Table 7.3 Perplexities of the language models that are used in the active learning experiments. Perplexities are calculated on the entire Hub5'00 evaluation set (SWB and CH combined). Partial words are included. For the random selection baseline, the median of three runs and the maximum deviation of the other runs is shown.

modelling, which could explain why gains from active learning are seen for decoding using *SWB+Fisher LM* but not using *specific LM*. Section 8.1 analyses the issue of selecting too many short utterances in more detail and proposes a solution.

7.4.2 80 hour results

For the 80 hour scenario, using the large SWB+Fisher language model NBest-BALD achieves absolute WER reductions of 11.1% and 12.5% (on SWB and CH) over the 20 hour system (30.3% $\rightarrow$ 19.2% and 46.7% $\rightarrow$ 34.2%). These absolute WER reductions are 7% and 8% higher than for the Random selection baseline as well as 3% and 6% higher than for the Confidence score approach.

For the 80 hour scenario, using the specific language model NBest-BALD achieves absolute WER reductions of 11.6% and 10.7% (on SWB and CH) over the 20 hour system (37.3% $\rightarrow$ 25.7% and 52.0% $\rightarrow$ 41.3%). These absolute WER reductions are 14% and 10% higher than for the Random selection baseline as well as 9% and 10% higher than for the Confidence score approach. They are also 9% and 6% higher than using the NBest-Ent approach. This clearly shows the benefit of using NBest-BALD for active data selection for this specific operating point. A statistical significance test was performed to compare the systems over the entire Hub5'00 evaluation set. The test is the Matched-Pair Sentence-Segment Word Error (MAPSSWE) statistical significance test (Gillick and Cox, 1989; Pallet et al., 1990). It showed that the p-values for the null hypotheses "there is no performance difference between NBest-BALD and the compared method" are below 0.001. It can also be seen that the specific language model that was used performs the best even though for the 40 hour data set, it performed poorer than the other approaches.

7.4.3 160 hour results

For the 160 hour scenario, using the large SWB+Fisher language model NBest-BALD achieves absolute WER reductions of 13.2% and 14.7% (on SWB and CH) over the 20 hour system (30.3% $\rightarrow$ 17.1% and 46.7% $\rightarrow$ 32.0%). These absolute WER reductions are 6% and 7% higher than for the Random selection baseline as well as 3% and 3% higher than for the Confidence score approach.

For the 160 hour scenario, using the specific language model NBest-BALD achieves absolute WER reductions of 14.3% and 13.8% (on SWB and CH) over the 20 hour system (37.3% $\rightarrow$ 23.0% and 52.0% $\rightarrow$ 38.2%). These absolute WER reductions are 8% and 8% higher than for the Random selection baseline as well as 5% and 3% higher than for the Confidence score approach. Here, the specific language model for NBest-BALD is also the best as measured by the perplexity. Interestingly, for the 160 hour scenario, all language models that are based on active learning outperform the Random baseline, while that was not the case for the 40 hour and 80 hour scenarios. This shows that all the active learning methods do not just select data that is useful for the acoustic model but also for the language model. However, at the same time, the utterances selected during the first batches are less useful for the language model than the utterances selected later on. The difference in performance between the NBest-BALD model and the NBest-Ent model was statistically significant at the $p < 0.01$ level as measured by the MAPSSWE and at the $p < 0.001$ level as compared to Random selection and Confidence scores.

For the 160 hour scenario, the differences in improvements are not as large as for the 80 hour scenario. The reason for this is likely that 160 hours is already more than 50% of the data set, and there are limitations on what even the most ideal active learning method could achieve as all "special" utterances with very high usefulness to training the ASR system have likey already been selected *i.e.* the potential for active learning to be excessively useful over Random selection should reduce as the size of the selected data increases. Clearly, for a 305 hour scenario, all methods will achieve identical WERs.

7.4.4 Additional results using stronger acoustic model

The previous results all used HMM-DNN hybrid ASR systems with a 7-layer fully connected HMM-DNN model that was trained using cross-entropy training. In this section, the results are based on HMM-DNN hybrid models that use factorised Time Delay Neural Networks (TDNNs),

	SWB	CH
Random	17.8%	32.1%
NBest-BALD	16.6%	29.8%

Table 7.4 Active Learning results on the Hub5'00 evaluation set (SWD/CH) for a total data set size of 80 hours. LFMMI-TDNN results with four-gram language model. Language models are trained using the selected data only. Comparable to the 80 hour, specific language model results in Table 7.2.

which were trained with the Lattice-Free Maximum Mutual Information (LF-MMI) objective (Povey et al., 2016). This was done by closely following the standard Kaldi recipes (Povey et al., 2011) but adjusting parameters to reflect the fact that only a quarter of the usual data is used for training. The neural network acoustic model had around 19 million parameters.

The two methods examined are Random selection and using NBest-BALD. For this, the utterances used for the previous 80-hour experiments are used for training *i.e.* the utterances are used to train an acoustic model with a different model structure to the model that was used for the selection. Here, the language model used for the decoding is the specific language model that was trained using the data that was used to train the acoustic model. This experiment examines whether or not the benefits of an actively selected data set are transferable from one model structure and learning algorithm to another. This is important because, in practice, engineers working on the data set might want to update the model or the training algorithm as speech recognition research evolves and because it might be more efficient to use a simpler model for the active data selection.

The model using the data from the Random selection had WERs of 17.8% for SWB and 32.1% for CH, which were 27.1% and 42.5% in Table 7.2. The model using the data from the NBest-BALD based selection had WERs of 16.6% for SWB and 29.8% for CH, which were 25.7% and 41.3% in Table 7.2. This demonstrates that the selected data is transferable from one model structure to the next and that NBest-BALD is able to select utterances that are useful for the ASR task in general. Based on the MAPSSWE the difference between the two systems was statistically significant at the $p < 0.001$ level.

7.5 Limitations

Section 7.4 showed that NBest-BALD can yield significant improvement over Random selection, Confidence score based selection and NBest-Entropy based selection. In the experiments, NBest-BALD was combined with a representativeness-encouraging approach that is based on the cosine-similarity of TF-IDF vectors (see Section 7.2.3). Both the calculation of the NBest-BALD score as well as the calculation of the cosine-similarity between each utterance pair are computationally expensive operations. For NBest-BALD eleven sets of decoding lattices as well as NBest-Lists were generated and stored. For the representativeness scores, the dot-product of all pairs of utterances was calculated which scales with the square of the utterances in the untranscribed dataset $\mathcal{D}_{ul}$. So whilst the improvements are statistically significant, it might be necessary to modify the method that was proposed in this chapter to fit a lower computational budget. This could include trying to reduce the number of dropout masks that are used or using a tighter decoding and re-scoring beam for faster NBest-BALD score calculation. It could also include reducing the size of the NBest-List. Furthermore, a more efficient representativeness-encouraging method could be explored.

7.6 Summary

This chapter introduced Bayesian active learning for ASR and specifically via NBest-BALD as a specific algorithm.

NBest-BALD selects utterances based on the mutual information between the prediction for the utterance and the model parameters θ, *i.e.* $\mathcal{I}[\theta, \mathbf{w}|\mathcal{D}_l, \mathbf{X}_i]$. Monte-Carlo Dropout is used to approximate sampling from the posterior of the model parameters. An N-Best list is used to approximate the hypothesis space such that the entropy of the output distribution can be calculated. NBest-BALD, as used in the experiments, was combined with a representativeness-encouraging approach that uses the dot-product of TF-IDF vectors as its similarity measure.

Experiments on the Switchboard corpus showed that NBest-BALD outperforms Random sampling and prior active learning methods that use Confidence scores or the NBest-Entropy as the informativeness measure. Especially when using an initial data set of 20 hour and selecting an additional 60 hour of data from a data set of 305 hours in total. Here, the absolute WER reduction due to the additional 60 hour of data were 14% and 10% higher than for random selection, 9% and 10% higher than for using confidence scores as well as 9% and 6% higher than using the NBest-Entropy. This was on the SWB and CH subsets of the Hub5'00 evaluation test set. As measured by the MAPSSWE the difference in the performance of these systems was statistically significant at the $p < 0.001$ level. Furthermore, the data selected by NBest-BALD also yielded the best language model as measured by the perplexity on the Hub5'00 evaluation test set.

Furthermore, the selected data was shown to be transferable to a different acoustic model structure. The original active data selection used a HMM-DNN hybrid model that was trained using frame-level cross-entropy training, and the results were transferable to an HMM-TDNN hybrid model trained using LF-MMI.

Chapter 8

Active learning: Utterance length normalisation and a Realistic cost function

The primary objective of this thesis chapter is to address two critical challenges in active learning for speech recognition.

The first challenge pertains to the tendency of active learning algorithms to select short utterances, which can negatively affect the performance of speech recognition models. This is because there is now a strong shift in the distribution used for training and for testing the Automatic Speech Recognition (ASR) model. Shorter utterances are over-represented in the selected data because shorter utterances have a higher variance in their active learning scores (see Figure 8.1). This results in a higher number of short utterances having high scores and therefore being selected by the algorithm (see Figure 8.2). To overcome this issue, a novel approach is proposed that involves sampling the length of the utterance from a histogram over the length of the utterance and then selecting an utterance from the corresponding bin. This method ensures that the selected utterances represent the distribution over the utterance lengths found in the data set.

The second challenge tackled in this thesis chapter is the fact that the cost function, which is used in most active learning experiments that are published for speech recognition, is not realistic. Typically, the cost of transcription is measured in terms of the number of hours or the number of utterances, which fails to accurately reflect the additional effort required for a transcriber to use contextual information to transcribe accurately. Transcribers must listen to neighbouring utterances to understand the context of a given utterance, making the process of transcribing an entire conversation or a chunk of neighbouring utterances more efficient than transcribing individual utterances. This thesis chapter proposes the use of a more realistic cost function that considers the need for transcribers to use contextual information. This cost function consists of the following components: a real-time factor for the time of the utterance that needs to be transcribed, a specified number of seconds before the start of an utterance that should be transcribed, a real-time factor for this specified time period, and an overhead in seconds that is required for any utterance. By incorporating this type of cost function into the active learning algorithm, the efficiency and effectiveness of speech recognition models can be significantly improved.

8.1 Histogram based sampling

8.1.1 Experimental Setup

The experiments in this section will largely follow the setup that was used Chapter 7 and was presented in Section 7.3. This leads to Tables 7.1 and 8.1 as well as Tables 7.2 and 8.2 containing some repeated Word Error Rates (WERs). It also leads to Tables 7.3 and 8.3 containing repeated perplexities.

To summarise the setup, the total pool of data that was used was a 305 hour subset taken from the Switchboard corpus such that there is no speaker overlap with the SwitchBoard (SWB) portion of the Hub5'00 test set that was used for evaluation. The initial data set that was used contained 20 hours of data. Two iterations of minibatch acquisition are run. For the first iteration 20 hours of data are selected and for the second 40 hours are selected. The models that are used are HMM-DNN systems built using HTK.

8.1.2 Motivation

The plot in Figure 8.1 depicts the relationship between the length of an utterance in terms of the number of words in its transcription and its corresponding informativeness, normalised by the utterance length. The normalised informativeness measure used here is one minus the average per-word confidence score. This is analogous to selecting utterances with the lowest average per-word confidence. However, for this analysis, a metric for which utterances with the highest scores are chosen is used. The model that is used to obtain the confidence-scores is the model that was trained on the initial 20 hour data set. The plot is a scatter plot, with each dot representing one utterance *i.e.* its length and informativeness. The utterance length is taken from the transcription based on confusion network decoding.

The x-axis of the scatter plot represents the utterance length in words, while the y-axis represents the informativeness of the utterance. The red line on the plot shows the average normalised informativeness at each length *i.e.* all normalised informativeness scores were

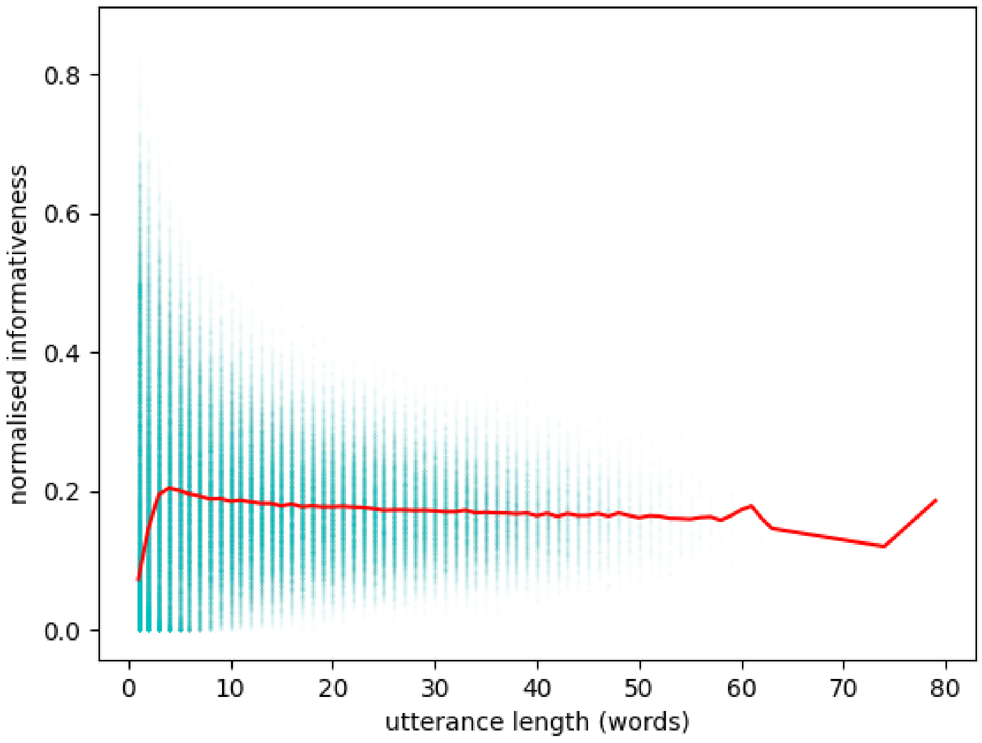
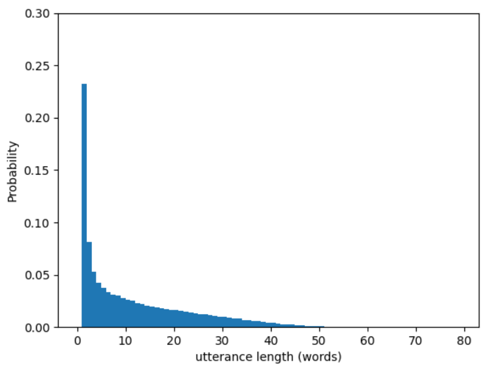


Figure 8.1 Scatterplot of the (normalised) informativeness using confidence scores. Each dot represents one utterance *i.e.* its normalised informativeness and its length in words. The red line represents the average at each word length.

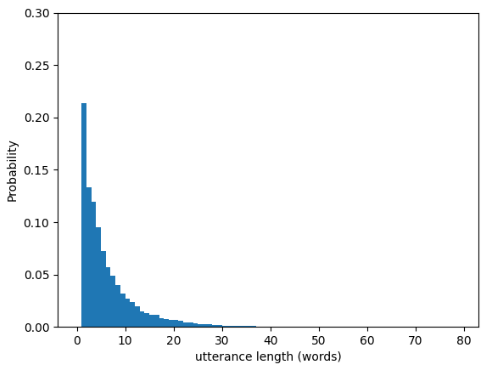
gathered for each utterance length and then averaged. It can be observed that the normalised informativeness is lower for short utterances with fewer than five words and then stabilises for the other lengths. Furthermore, the plot shows that the variance in informativeness is higher for shorter utterances and lower for longer utterances.

The plot in Figure 8.1 has important implications for the selection of utterances in active learning. Commonly, and the procedure followed in Chapter 7, the selection criterion for the selected data set is based on a threshold of normalised informativeness per utterance. In this case, the fact that the scores of shorter utterances have a higher variance means that short utterances will be over-represented in the data that is selected for annotation as compared to longer utterances. This could lead to a bias in the model towards shorter utterances and result in sub-optimal performance.

The effect of selecting shorter utterances in active learning can be seen in Figure 8.2, which presents two histograms for two different data sets. The x-axis of both histograms represents the length of utterances in words, while the y-axis represents the proportion of utterances of



(a) Entire data set



(b) Selected data

Figure 8.2 Histogram of the utterance lengths for the entire data set and the selected data. The selected data uses confidence scores for selection. The selection is done by thresholding the informativeness scores from Figure 8.1.

that length. Figure 8.2a shows the histogram for the entire untranscribed data set. Figure 8.2b shows the histogram for a 20 hour data set that was selected using active learning based on confidence scores. This is exactly the data set that was used (together with the initial 20 hour data set) to train the Confidence Score model in Table 7.1 that had a WERs of 31.6% and 47.2% on the SWB and CH portions of Hub5'00.

The comparison of the two histograms reveals that Figure 8.2a has much more of the probability mass placed on longer utterances, while Figure 8.2b has a higher proportion of shorter utterances. This provides further evidence of the bias and shift in length distribution towards selecting shorter utterances in active learning when using the thresholding of normalised per utterance informativeness as the method for selection.

To address the bias towards selecting shorter utterances in active learning, a method will be introduced that ensures that the histograms of the entire data set and the selected data set are consistent.

8.1.3 Method

The method for addressing the bias towards selecting shorter utterances in active learning for ASR starts by calculating the active learning scores for the utterances in the same way as usual. Any active learning method can be used for this purpose. Once the active learning scores are calculated, a histogram is constructed over the utterance lengths in the same way as shown in Figure 8.2a. In this histogram, nearly all utterance lengths (in words) have one histogram bin. Rarer utterance lengths, with a count of below 1000, are grouped together into bins with a count of just over 1000. For Switchboard, these are utterances that are longer than 40 words. The utterances, together with their active learning score, are then divided into these histogram bins. This grouping was used so that the active learning scores still have an impact for rarer utterances; in the extreme case of selecting an utterance length that has a count of one, the informativeness measure is not used.

To generate the selected data set, an utterance length is first sampled from the histogram, which is used as a categorical distribution over the utterance length. Then, the top-scoring utterance in terms of the active learning score is selected within the histogram bin belonging to the sampled word length. This process is repeated until the active learning budget is exhausted.

This method ensures that the selection of data is not biased towards shorter utterances. By sampling the word length first and then selecting the top-scoring utterance within the corresponding histogram bin, the selected data set will be expected to have the same distribution of utterance lengths as the entire data set (*i.e.* the distribution of Figure 8.2a). This approach helps to ensure that the active learning model is trained on a representative subset of the data in terms of the utterance length. The proposed method is also virtually hyperparameter-free (except for the minimum bin size).

In active learning, hyperparameter optimisation (such as for the β in the information density framework - see Equation (5.10)) is difficult. For many learning scenarios, such as supervised or semi-supervised learning, hyperparameter optimisation is done on a held-out development test set. During training many hyperparameter settings can be tried and the one that achieves the best performance on the held-out development set is selected. This approach is commonly used in deep learning, where hyperparameters such as the learning rate, the number of layers, and the width of each layer can significantly impact the performance of the model. The held-out development set provides a way to evaluate the performance of the model under different hyperparameter settings and enables the selection of the best-performing model for deployment. However, in active learning, the process of selecting additional data for labelling cannot be done more than once; hence hyperparameter optimisation is more challenging. Active learning can be simulated on an already labelled data set. That data set would be either a smaller data set or a data set of similar size from a different task. However, it is unclear how well hyperparameters will translate from a smaller to a larger data set or from one task to another.

The computational complexity of this selection method is proportional to the number of bins in the histogram times the number of utterances. The number of bins is low, and hence this method is very efficient, especially in comparison to many representativeness-based methods that scale with the square of the number of utterances (see Equation (5.10)).

The downside of using the histogram-based sampling is that it is a hard constraint on the selected data. If the model is already perfect at decoding single-word utterances, it would not be useful to decode them. In this case, trying to use a method that has a soft constraint on the length might be more beneficial.

8.1.4 Experimental Results

In the results section, three tables are presented to show the results of the active learning data selection. Tables 8.1 and 8.2 compare active learning using confidence scores as the informativeness measure to random selection and to the proposed histogram-based sampling method, which also uses confidence scores. The results in Table 8.1 are obtained via decoding with a large language model trained on data from both the Switchboard and the Fisher corpus. The results in Table 8.2 are obtained via decoding with a language that was trained on only the transcribed data specific to the active learning result *i.e.* the same data that was used for training the acoustic model.

	SWB+Fisher LM		
	20h	40h	80h
Random	30.3 / 46.7	24.3±0.4/ 40.1±0.6	19.9±0.4 / 35.1±0.2
Conf Score	-	23.6 / 40.6	19.5 / 34.9
Conf Score + histogram	-	22.8 / 38.9	18.8 / 33.9

Table 8.1 Active Learning results on the 2000 HUB5 evaluation set (SWD/CH). HMM-DNN acoustic models, tri-gram language model with Confusion Network Decoding. Language model was trained on SWBD plus Fisher data.

	Specific LM		
	20h	40h	80h
Random	37.3 / 52.0	31.4±0.2 / 47.4±0.4	27.1±0.4 / 42.3±0.2
Conf Score	-	31.6 / 47.2	26.7 / 42.3
Conf Score + histogram	-	30.1 / 44.8	25.8 / 40.1

Table 8.2 Active Learning results on the Hub5'00 evaluation set (SWD/CH). HMM-DNN acoustic models, tri-gram language model with Confusion Network Decoding. Language models are trained the selected data only (specific LM).

Language model perplexities of the specific language models and the large SWB+Fisher language model are presented in Table 8.3. As before, they are calculated on the entire Hub5'00 data set.

	Language models		
	20 h	40 h	80 h
Random	129.8	115.3±0.3	105.2±0.1
Conf Score	-	115.2	105.5
Conf Score + histogram	-	113.7	104.3
Full Train Set		90.6	
SWB+FSH		78.6	

Table 8.3 Perplexities of the language models that are used in the active learning experiments. Perplexities are calculated on the entire Hub5'00 evaluation set (SWB and CallHome (CH) combined). Partial words are included. For the random selection baseline the median of three runs and the maximum deviation of the other runs is shown.

For the 40 hour scenario, using the large SWB+Fisher language model, the histogram-based sampling achieves absolute WER reductions of 7.5% and 7.8% (on SWB and CH) over the 20 hour system (30.3% $\rightarrow$ 22.8% and 46.7% $\rightarrow$ 38.9%). These absolute WER reductions are 25% and 18% higher relative to the Random selection baseline as well as 12% and 28% higher than for the Confidence Score approach.

For the 40 hour scenario, using the specific language model the histogram-based sampling achieves absolute WER reductions of 7.2% and 7.2% (on SWB and CH) over then 20 hour system (37.3% $\rightarrow$ 30.1% and 52.0% $\rightarrow$ 44.8%). These absolute WER reductions are 22% and 57% higher than for the Random baseline as well as 26% and 50% higher than the Confidence Score approach. These improvements were statistically significant as measured by the Matched-Pair Sentence-Segment Word Error (MAPSSWE). Based on Table 8.3, it is clear that a large part of this improvement is due to a stronger language model. The language model perplexity is 1.5 points lower as compared to not using the histogram-based sampling. Comparing these 1.5 points of improvement to the differences in perplexity observed in Table 7.3 and to the perplexity deviation from multiple random selections, this is significant. This improvement likely stems from longer utterances being more diverse, which can be seen from the fact that the specific language model for the histogram-based sampling contains 14% more tri-grams relative to the purely Confidence score selection.

For the 80 hour scenario, using the large SWB+Fisher language model, the histogram-based sampling achieves absolute WER reductions of 11.5% and 12.8% (on SWB and CH) over the 20 hour system (30.3% $\rightarrow$ 18.8% and 46.7% $\rightarrow$ 33.9%). These absolute WER reductions are 11% and 8% higher than for the Random selection baseline as well as 6% and 10% higher than for the Confidence score approach.

For the 80 hour scenario, using the large specific language model, the histogram-based sampling achieves absolute WER reductions of 11.5% and 11.9% (on SWB and CH) over the 20 hour system. These absolute WER reductions are 13% and 23% higher than for the Random selection baseline as well as 8% and 11% higher than for the Confidence Score approach.

The results indicate that histogram-based sampling provides a significant boost in performance. The largest improvement is for the 40 hour scenario when using the specific language model. Here the improvement over the random selection baseline is 20% and 57%. In Chapter 7, the largest improvements were seen for the 80 hour scenario. The results in Chapter 7 had the issue that the initial utterances that are chosen are less useful due to the problem addressed in this section. Given that this is solved in this section, the operating point for the largest improvement moves to a smaller data size. The smaller the to-be-selected data set is relative to the unlabelled data set, the better active learning should perform. This is because more "special" utterances are available for selection that could have a disproportionate effect on the performance of the ASR system.

Furthermore, these results show that active learning, in general, also selects data that is more useful for the language model than Random selection. Previously, without the use of histogram-based sampling, the perplexities in Table 8.3 might suggest that data is selected to improve the acoustic model but not the language model. However, once the distribution over the utterance length is matched, the selected data significantly improves the language model.

8.1.5 Discussion with Related Work

The proposed histogram-based sampling method falls under the category of representativeness-based methods (see Section 5.4.4). It selects a subset of utterances that is representative in terms of their distribution over the lengths of the utterances. Prior art relies on computing a score for representativeness and possibly multiplying it with an informativeness score (as in the information density framework (Settles and Craven, 2008)). In the proposed method, representativeness is achieved directly by ensuring that the desired distribution (here, over the lengths of the utterances) is consistent.

Other representativeness-based methods often use similarity scores that are computed for all pairs of utterances, which can be computationally expensive, especially when dealing with large data sets. In contrast, the histogram-based sampling method has much lower computational complexity. It avoids the need to compute similarity scores, and directly samples from the histogram, resulting in a more efficient selection process. The proposed method is, therefore, a novel and efficient approach to representativeness-based active learning. It also does not have any hyperparameters to tune.

The histogram-based sampling method is not limited to using only the length of an utterance as the feature for representing the distribution. Other features of an utterance can also be used to construct the histogram, and thus, the method can be applied to any feature of an utterance that is relevant to the task at hand. A continuous valued feature such as the percentage of non-speech in an utterance could also be used.

Furthermore, the proposed method has potential benefits when there is a mismatch between the unlabelled data set available for active learning and the target domain. If the histogram is derived from a development set, the histogram-based sampling could bridge the gap between the train and test sets, making the active learning selection highly effective. By constructing a histogram based on the relevant features of the development set, the proposed method can help to ensure that the selected data is more representative of the test set, ultimately leading to better performance on the test set.

8.2 Selection of Consecutive Utterances

The following will build a model for the labelling cost of transcription in a conversation or a meeting. The labelling cost function takes into account the conversational structure of a data set, recognising that utterances are not independent and can occur in a conversational sequence. A group of utterances that is selected for transcription and occurs consecutively is referred to as a “chunk” of utterances. The cost function takes into account that labelling a chunk of utterances that occur consecutively takes less time than randomly selected utterances that have the same total length. This approach allows for a more accurate estimate of the cost of transcription, taking into account the contextual relationships between utterances and their associated transcription costs. Ultimately, the labelling cost model provides a more accurate estimate of the cost of active learning. Previous publications (and previous sections in this thesis) used the total duration of the selected data set (in seconds) as the transcription cost or even the number of selected utterances.

8.2.1 Defining labelling cost for consecutive utterances

Section 5.2.3 gave a detailed account of the labelling cost involved in the transcription of speech. The cost of labelling an utterance is determined by its duration and the desired level of transcription accuracy. Depending on the desired accuracy, the duration is multiplied by a Real-Time Factor (RTF), which can range from close to 1 for closed-captions to 50 for very careful and detailed transcriptions. For the QuickTranscription standard that was used to transcribe the Fisher corpus (Cieri et al., 2004b) the RTF was around 6. It is also the case that the transcriber might need to listen to the surrounding context in order to label an utterance. This leads to a certain overhead per chunk of consecutive utterances. There might be even more overhead per chunk of utterances, such as the time required for starting a certain user interface.

In the model for labelling cost that is proposed in this model, after the automatic segmentation, a certain gap (t_1, say one second) is left between consecutive utterances, which contributes to an overhead cost of transcription per utterance. The duration of the utterance T is multiplied by a Real-Time Factor RTF. Additionally, for each chunk of utterances, there is an overhead cost of (t_2), which accounts for factors such as listening to surrounding utterances and using the user interface. Smaller values of RTF and larger values t_2 would favour the selection of longer chunks of utterances. If, after selection, two chunks overlap or are adjacent, the cost is automatically adjusted. The selection of a chunk of N consecutive utterances with a total length of T, thus has a cost of:

$$\text{cost} = T * \texttt{RTF} + (N-1) \cdot t_1 + t_2 \tag{8.1}$$

For the experiments, $t_1 = 1.0\ s$ and $\texttt{RTF} = 6.0$ is used throughout. Different overheads t_2 are examined in Figures 8.3, 8.4 and 8.5.

8.2.2 Selecting at different levels of granularity

The labelling cost model in Equation (8.1) can be used to examine the effect of selecting utterances at different levels of granularity in active learning. The model can be applied to modelling the cost of labelling individual utterances, whole conversation sides, or chunks of utterances that have a fixed or variable size. By choosing larger chunks, the selection process cannot exploit the variation of informativeness of the utterances as well as when choosing individual utterances, but at the same time, the relative cost of the overhead is less. For the selection methods, the informativeness of any amount (utterance, chunk, side) of data is defined as the sum of one minus the per-word confidence score.

For individual utterances, the active learning utterance selection process is similar to previous cases (in this thesis), but the cost is increased by the overhead that exists for each utterance (which is now a chunk). The overhead represents a large portion of the cost; thus, a smaller total data set size will be selected. Furthermore, utterances are selected based on the informativeness per unit cost; thus, for large overheads, t_2 longer utterances will tend to be selected as the relative cost-increase from the overhead is lower for them.

For the selection of whole conversation sides, the informativeness for each side is calculated, and then the sides with the highest informativeness per unit cost are selected. The overhead is not a significant factor in the selection of whole sides because it is small relative to the length of a side.

For fixed-sized (in terms of the number of utterances) chunks, the process is similar to the selection of sides. The informativeness is calculated for all possible chunks of a specific size, and those chunks with the highest scores are selected. The larger the size of the chunk, the smaller the effect of the overhead.

For variable-sized chunks, all possible chunks from size one (a single utterance) to the size of a side are considered. The informativeness is calculated in the same way as for the other methods. If the overhead is low (or zero) the selection will select mostly individual utterances. For a very high overhead, the selection will be most similar to selecting sides.

8.2.3 Experiments

The experiments in this section will largely follow the setup that was used Chapter 7 and was presented in Section 7.3. The same initial data set of 20 hours and associated ASR-model was used. One iteration of batch acquisition was run to select 40 hours of data, leading to 60 hours of supervised data.

The labelling cost function (Equation (8.1)) used RTF of $\texttt{RTF} = 6.0$, a gap between utterances of $t_1 = 1.0$ seconds, and an overhead of $t_2 = 30$ seconds, which corresponds roughly to the Quicktranscriptions standard used in the Fisher data collection and is coherent with my own self-experimentation for transcribing audio. $t_2 = 30$ was also chosen because it was not only realistic but also an interesting operating point based on Figure 8.3.

8.2.3.1 Effect of the overhead

Figures 8.3, 8.4 and 8.5 show the results of this selection method. Here, a real-time factor of $\text{RTF} = 6$ is used when selecting data with a budget of 240 h.

Figure 8.3 shows how the total informativeness of the selected data set (the sum of one minus the confidence scores) varies with the overhead t_2. Select ing individual utterances yields the highest informativeness for an overhead of 0 s, and selecting sides yields the highest informativeness for an overhead of 60 s. Selecting using variable-length chunks always yields the highest total informativeness.

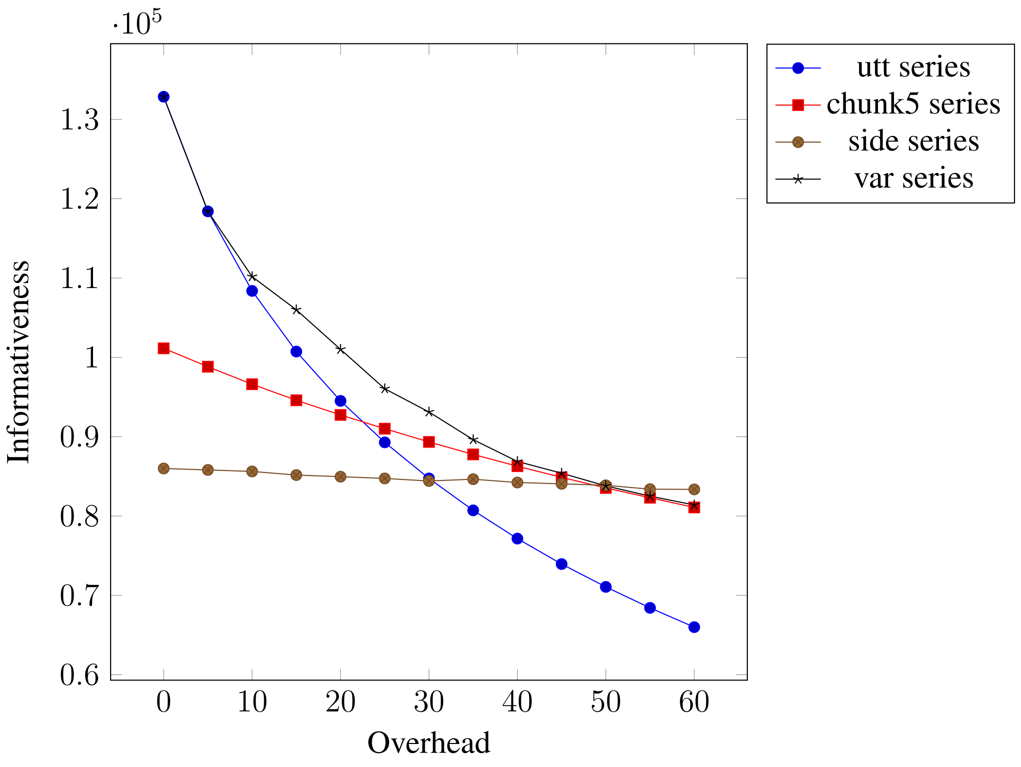


Figure 8.3 Graph of (total) Informativeness vs Overhead. The chosen real-time factor is 6. Confidence scores are used for the informativeness.

Out of all the chunk sizes, 5 yielded the highest informativeness for an overhead of 30 s, which is the operating point that will be used during the active learning experiments. It was previously explained that hyperparameter optimisation is generally not possible for active learning; however, here, optimising the chunk size does not involve (simulating) the transcription of any data. Hence, this is perfectly valid.

Overall Figure 8.3 demonstrates the need for taking the sequential structure of conversations or meetings into account when doing active learning for these types of data sets. If informativeness were perfectly correlated with WER reduction, the variable-length chunk-level selection would give the best results.

Figure 8.4 shows how the size of the selected data set (in seconds) varies with the overhead. It demonstrates how quickly the amount of data that is selected decreases for the various approaches. When the overhead t_2 is zero, utterance-level selection selects more data than the other approaches because t_1 is not a cost to it.

Figure 8.5 is a plot that shows the relationship between the informativeness and the total data size. It shows that for a certain number of seconds, the smaller chunk size always has the higher informativeness. It also shows how the variable chunk size algorithm varies from being close to selecting individual utterances to being more similar to the fixed chunk size of 5.

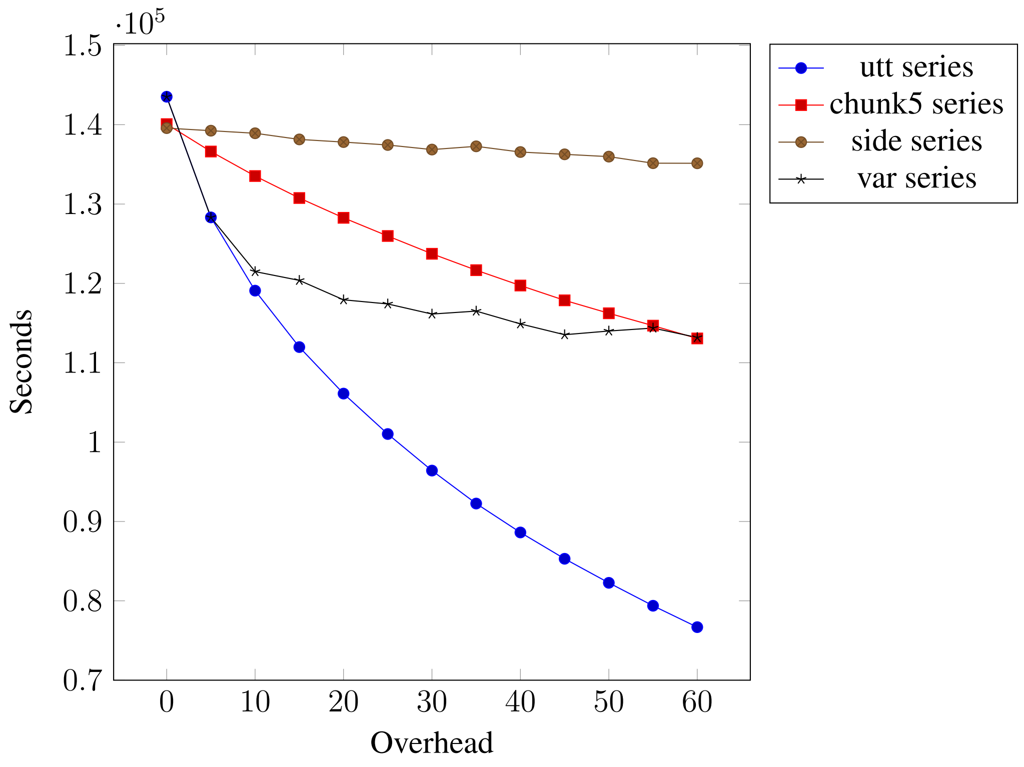


Figure 8.4 Graph of size of the selected data set in seconds vs Overhead. The chosen real-time factor is 6. Confidence scores are used for the informativeness.

8.2.3.2 Data selection experiments

The active learning process had a labelling budget of 240 hours, aimed at selecting $240/\texttt{RTF} = 40$ hours of data, which is largely the case for the side-based selection where the cost due to the overhead is negligible. ASR-models trained on data that was selected at the different levels of granularity that are shown in Figure 8.3 were compared.

Table 8.4 shows the results for the active learning process. The selection method on the utterance level gives the highest WER for both evaluation sets using both language models. This is likely due to the fact that the method selects the least amount of data (as measured in hours). The total training set size (including the 20 h seed data set) is 46.8 h. For the chunk-level selection, the side-level selection and the variable-length chunk-level selection, the resulting training set sizes are 54.4 h, 58.0 h and 52.3 h, respectively. The fact that the utterance-level selection gave the highest WER clearly demonstrates the need for using a labelling cost function that takes into account the conversational structure of a data set as well as a selection method that can properly use this kind of cost function.

When using the large SWB+Fisher language model, the lowest WERs were achieved by the chunk-level selection for chunks of size five slightly lower than using the variable-length chunk-

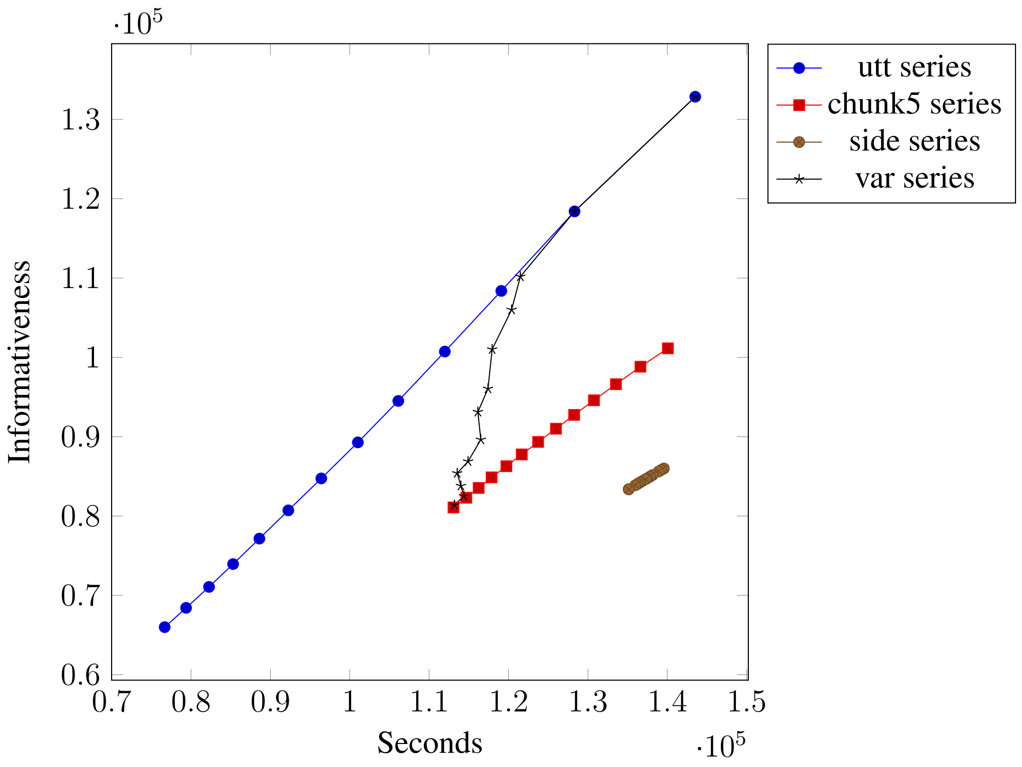


Figure 8.5 Graph of (total) Informativeness vs size of the selected data set in seconds. The chosen real-time factor is 6. Confidence scores are used for the informativeness.

level selection. When using the specific language model, the lowest WERs were achieved by the side-level selection. These results thus show that the informativeness as shown in Figure 8.3 is not the ideal proxy for the final performance given that the variable-length chunk-level selection does not outperform the other methods. Based on the MAPSSWE the differences in performance between the utterance-level selection and the other methods were statistically significant at the $p < 0.001$ level. The differences in performance between the variable-length chunk-level selection and the fixed-length chunk-level selection were not statistically significant.

8.3 Summary

This chapter focused on two issues of current active learning methods. The first focus was on the fact that active learning often does not perform well for the acquisition of the first batch (if the batch is small). The second focus was on the fact that previous active learning publications regarded utterances as completely independent and did not take into account the reduced cost of transcribing utterances that occur in a sequence.

Granularity	SWB+Fisher LM	Specific LM	Data (hours)
Utterance	24.2 / 40.1	31.3 / 46.7	46.8
Chunk-5	21.6 / 37.0	29.4 / 45.1	54.4
Side	22.1 / 37.7	29.2 / 44.7	58.0
Variable Chunk	21.7 / 37.2	29.7 / 45.5	52.3

Table 8.4 Active Learning results for multiple levels of selection granularity on the Hub5'00 evaluation set (SWD/CH). HMM-DNN acoustic models, tri-gram language model with Confusion Network Decoding. Language model was trained on SWBD plus Fisher data or the specific language model trained on only the selected data. The seed model was trained on 20 h of data. It was aimed to select another 40 h of data. The labelling budget was $40 \cdot \texttt{RTF} = 240h$. Due to overhead costs (t_1 and t_2), the total data size for training is less than 60 h.

In Section 8.1, a novel method for encouraging representativeness in active data selection was developed. The method first builds a histogram over the lengths of the utterances. Then, to select an utterance, a word length is sampled from the histogram and the utterance with the highest informativeness within the corresponding histogram bin is chosen. This ensures that the selected data set has a similar distribution of utterance lengths to the overall data set. Experiments on the Switchboard corpus showed that this method significantly improves the performance of active learning for the first batch. For the operating point with the highest improvement, the absolute WER reductions due to an additional 20 hours of data were 22% and 57% higher than for the random selection baseline and 26% and 50% higher than for the confidence score approach. This was on the SWB and CH subsets of the Hub5'00 evaluation test set. For this operating point, an initial data set of 20 hours was used and an additional 20 hours of data was selected. The issue that active learning does not perform well for the first batch is a long-standing issue in active learning for ASR, which was first reported by Hakkani-Tür et al. (2002) and more recently by Radmard (2022) as well as in Chapter 7 of this thesis. The proposed histogram-based sampling method clearly addresses this issue.

In Section 8.2, first, a definition of a cost function was introduced which takes into account the sequential nature of conversations and meetings. Second, it was examined at which level of granularity data should be selected given the new cost function. Selecting data on the utterance-level, as fixed-length chunks of consecutive utterances, as variable-length chunks of consecutive utterances and on the side-level were examined. The cost function combines a RTF for the utterance length (in seconds) with an overhead for each utterance (t_1) and an overhead for a chunk of consecutive utterances (t_2). The overhead t_2 affects the utterance-level selection methods (which the methods in literature rely on) the most and this method yielded the worst ASR performance. This result showed that it is crucial to focus on methods for selection that can take a better cost function into account. Whilst the highest informativeness of the selected data set was achieved by the variable-length chunk-level selection, the lowest WERs were achieved by the fixed-length chunk-level selection and the side-level selection.

Chapter 9

Improved tokenisation methods for self-supervised learning

Section 4.8 introduced methods for unsupervised representation learning for Automatic Speech Recognition (ASR) based on self-supervised learning algorithms such as wav2vec 2.0 (Baevski et al., 2020b) or Hidden unit BERT (HuBERT) (Hsu et al., 2021a). Self-supervised learning for speech and audio (Mohamed et al., 2022) is about training an audio encoder that learns representations of audio by being trained to predict targets that are derived from the input audio itself. Such tasks include learning to predict multiple feature encodings of the input (Pascual et al., 2019; Ravanelli et al., 2020), distinguishing near-by future speech features from temporally distant ones (Oord et al., 2018; Schneider et al., 2019) or directly predicting future frames (Chung et al., 2019), and prediction of audio features underneath a mask (Baevski et al., 2020a,b; Hsu et al., 2021a,b). These self-supervised algorithms can leverage large amounts of unlabelled data effectively.

Many modern self-supervised learning algorithms are based on Masked-Prediction Pre-Training (MPPT), where portions of the inputs are masked, and the neural network is trained to predict labels for the masked portions. These labels are derived in an unsupervised fashion. In wav2vec2.0 (Baevski et al., 2020b) and data2vec (Baevski et al., 2022), the labels are real-valued. In HuBERT (Hsu et al., 2021a) and WavLM (Chen et al., 2022) the labels are discrete.

In ASR, after an encoder model is pre-trained using one of these algorithms on the unlabelled data, the model is finetuned using labelled data. Usually, the amount of unlabelled data is much larger than the amount of labelled data. The first part of this chapter will focus on using this small amount of labelled data to improve the targets that are usually derived in an unsupervised manner. This will be referred to as *biased* self-supervised training as the targets will be biased towards the supervised task. Furthermore, by treating the biased tokens as discovered phonetic units, models can be pre-trained by predicting these tokens directly, even for unmasked audio. This chapter proposes that this style of training, which is akin to cross-entropy training of DNN-HMM hybrid models (Bourlard and Morgan, 1994; Dahl et al., 2012), can be highly effective for training streaming ASR models with a restricted input window.

The discrete labels in HuBERT and WavLM are derived using frame-level clustering using K-Means clustering (Lloyd, 1982), which provides the tokenisation of the input sequence that is used for training. In the second part of this chapter, a sequence-level tokenisation based on Hidden Markov Models (HMMs) will be used.

9.1 Masked-Prediction Pre-Training

9.1.1 Pre-Training

This section describes MPPT by expanding on the description of HuBERT in Section 4.8.2.3. The final training method overlaps with the biased self-supervised training, and the terminology introduced in this section will be relevant when biased self-supervised training is outlined.

MPPT algorithms contain three main parts. A model structure that has an input and an output sequence, a method to obtain labels for each output step and a way to train using this label sequence.

In Hsu et al. (2021a), the model structure uses a convolutional encoder to encode the raw waveform input. This sequence has an output frequency of 50 Hz. This is followed by a convolutional layer with a large kernel size that is used for positional encodings. This is followed by multiple (*e.g.* 12 in HuBERT-Base) Transformer layers (Vaswani et al., 2017). The output embeddings from the final Transformer layer are then used to predict the underlying token label using a projection layer followed by the output layer. This chapter uses a modified and more efficient model structure that is explained in detail in Section 9.3.2.

The label sequence that is used to train the HuBERT model is obtained using K-Means clustering. K-Means clustering is used to partition a data set into K clusters based on similarities between data points. The algorithm initialises K cluster centres randomly, and then assigns each data point to the closest cluster centre based on the Euclidean distance between the data point and each cluster centre. The algorithm then updates the cluster centres by computing the mean of all data points assigned to each cluster and the process is repeated until the cluster assignments no longer change or a maximum number of iterations is reached. This algorithm has similarities to the Expectation Maximisation (EM)-algorithm in Section 3.9.1.1. The assignment of data points is similar to the E-step, but with hard assignments instead of using a probability distribution. Updating the cluster centres is similar to the M-step, where the parameters are updated to maximise the objective function. To obtain the label sequence, clustering is performed globally by using the entire corpus (or a representative sample of it) and not locally for each sequence.

In MPPT, as suggested by the name, the model is trained by masking portions of the input and then predicting the labels that lie underneath the masked inputs. In Hsu et al. (2021a), the output of the convolutional encoder is masked instead of the raw input sequence. HuBERT training occurs in multiple iterations. The first iteration of training uses a target sequence obtained by clustering speech features (Mel-scale Frequency Cepstral Coefficients (MFCCs)). For the succeeding iterations, hidden embeddings from the trained model of the previous iteration are clustered. The correct layer to use for this can be found based on the metrics described in Section 9.1.3. In order to model the evolution of the speech frames, as needed in the first iteration of training, the model has to learn and represent the underlying phones. Hence, the tokenisation that is used for the second iteration of training has a stronger correlation with the underlying phones.

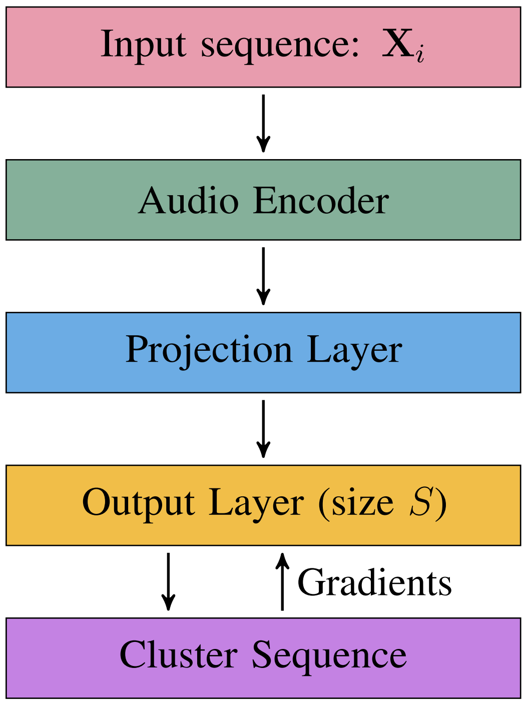
The unsupervised training pipeline is outlined in Algorithm 1, including the final fine-tuning step. This pipeline is equivalent to the HuBERT-Base training in Hsu et al. (2021a).

Algorithm 1 The (unbiased) masked-prediction pre-training and fine-tuning.

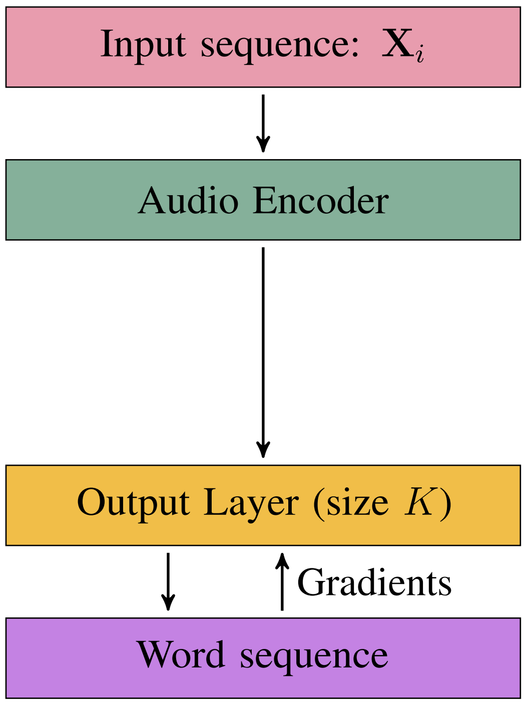
```
Let $I$ be the number of iterations
Let $\mathcal{D}_l$ be the labelled data set
Let $\mathcal{D}_{ul}$ be the unlabelled
Set $i \leftarrow 0$
**while** $i < I$ **do**
    Decide which vectors to cluster. MFCC features for $i = 0$ otherwise choose layer of $\mathcal{M}_{i-1}$ based on label purity.
    Cluster $\mathcal{D}_{ul}$ to obtain tokenisation $\mathcal{T}$
    Randomly initialise the model $\mathcal{M}_i$
    Train $\mathcal{M}_i$ on $(\mathcal{D}_{ul}, \mathcal{T})$
**end while**
Set $\mathcal{M}_I$ to be $\mathcal{M}_{I-1}$ with replaced output layer (see Figure 9.1b)
Finetune $\mathcal{M}_I$ on $\mathcal{D}_l$
**return** $\mathcal{M}_I$
```

9.1.2 Fine-tuning

After pre-training, the trained HuBERT model can be finetuned. This means that it is incorporated into a full ASR system (*e.g.* Connectionist Temporal Classification (CTC) or Transducer models) and then the whole system is trained on a supervised ASR data set. For this ASR training stage, the encoder model's projection layer and output layer are removed. They are



(a) Pre-training



(b) Fine-tuning

Figure 9.1 The model during the pre-training stage and during the fine-tuning stage. During self-supervised pre-training the goal is to pre-train the 'Audio Encoder'. For a certain number of updates at the start of fine-tuning (10k in the experiments of this chapter) the weights of the Audio Encoder are frozen and only the output layer is trained.

replaced with a Softmax layer that maps to the set of ASR system output tokens. This is shown in Figure 9.1, where Figure 9.1a shows the audio encoder model during pre-training and Figure 9.1b shows the audio encoder model during finetuning. In order to ensure that the gradients that are calculated using the randomly initialised output layer do not destroy the learned parameter setting in the pre-trained audio encoder model, the pre-trained audio encoder usually remains frozen for a certain number of updates (*e.g.* the first 10% of training), after which the encoder is also updated.

9.1.3 Evaluation Metrics

The training and evaluation pipeline, including pre-training, fine-tuning and decoding, has many stages. Hence, the final Word Error Rate (WER) should not be the only metric to analyse these kinds of audio encoder models. Statistics regarding the accuracy of the trained model on the masked-prediction pre-training task or the correlation between clusters and ground truth labels give insight into how the pre-training is performing.

Masked Prediction Accuracy This is the accuracy that the audio encoder model achieves on the masked-prediction pre-training task. For the same label sequence, this metric is correlated with final WER *i.e.* a higher accuracy leads to a lower WER. For the same model architecture and the same number of training steps, this metric is an indication of how “easy" the task is. The task can be made easier simply by reducing the number of clusters K. It can also be easier because the tokenisation is more consistent. This increased consistency can be viewed to be analogous to having less label noise in supervised training. Thus, this would be desirable.

Cluster-Purity and Label-purity These two metrics are calculated based on labelling each frame with its corresponding cluster $c_t \in [1, \dots, K]$ and with a ground-truth label $l_t \in [1, \dots, L]$, which could be phones or clustered (bi-)characters. This labelling should be done on a held-out development set. From this alignment, a joint count (count(c_t, l_t)) can be calculated over the entire data set, which is the basis for these metrics. Cluster-purity is the maximum achievable accuracy for predicting the cluster purely based on the label:

$$\frac{1}{T}\sum_{l=1}^{L}\max_{c}\left[\text{count}(c,l)\right] \tag{9.1}$$

where T is the number of frames in the entire data set that is being used for the calculation (*e.g.* the development set). Label-purity is the maximum achievable accuracy for predicting the ground-truth label purely based on the cluster.

$$\frac{1}{T}\sum_{c=1}^{K}\max_{l}\left[\text{count}(c,l)\right] \tag{9.2}$$

These metrics indicate how much the cluster tokens are correlated with the ground-truth labels and therefore they give an indication towards how similar the masked-prediction pre-training is to ASR training. Hence, these metrics are highly correlated with final decoding WER. It is important to note that when increasing K for the same embeddings, the cluster-purity will go down, and the label-purity will go up. Therefore these statistics are difficult to compare for models with different K.

WER recovery The WER recovery rate (WRR) is a common metric used to evaluate semi-supervised learning algorithms. It measures the ratio between the absolute improvement in WER from semi-supervised learning and from (oracle) supervised learning. In academic experiments the data sets used for semi-supervised learning are often created by taking a fully-supervised data set and treating a portion of it as “unlabelled”. Oracle supervised learning refers to training on the unlabelled data as if it were labelled. This provides a lower-bound for the WER when trained in a semi-supervised mode.

$$\mathrm{WRR} = \frac{\mathrm{WER}_{\mathrm{super}} - \mathrm{WER}_{\mathrm{semisup}}}{\mathrm{WER}_{\mathrm{super}} - \mathrm{WER}_{\mathrm{oracle}}} \tag{9.3}$$

9.2 Biased Masked Prediction Pre-Training

Although HuBERT demonstrates competitive WERs after fine-tuning, there are some problems that cannot be overcome without any supervised signal. Good ASR performance is not only about obtaining a concise representation of the audio but also about learning which information to ignore (*e.g.* speaker or channel information) and which to keep (*e.g.* phonetic information). The objective of MPPT allows the model to learn to ignore some variation in the data (the intra-cluster variation in comparison to fully reconstruction-based methods), but it is likely that all speaker-related information is kept. The cluster tokens that are obtained for training can represent various phones. However, they represent phones in various contexts in terms of the channel and the speaker. The reason for this is that clustering is challenging and inherently ambiguous when samples of different clusters are not well separated in the feature space. Without a supervision signal, clustering is not well-defined. Having effectively too many targets leads to the masked-prediction pre-training task being much more difficult than it would otherwise need to be, which, in turn, leads to slow training. Current self-supervised learning pipelines that are based on MPPT are often trained for a very large number of updates, for example Hsu et al. (2021a); Wang et al. (2022) train for 400k updates and Chen et al. (2022) trains for 1 million updates.

This section introduces a supervised signal into the pipeline described in Algorithm 1 of Section 9.1.1. The first iteration of training is identical to the standard unbiased masked-prediction pre-training. The difference to the unbiased masked-prediction pre-training training comes before the second iteration of masked-prediction pre-training. To bias the masked-prediction pre-training towards the desired task, the trained model of the first iteration is fine-tuned using a supervised loss for a small number of updates. This biases the model and, thus, the clusters towards the supervised task. Whilst the later experiments use a CTC (Graves et al., 2006) loss to make the masked-prediction pre-training ASR specific, this biasing step could be done using any speech processing task or even multi-task training. After the biasing step, the embeddings are again clustered using K-Means clustering. The results in Section 9.3.9 will later show that compared to clustering unbiased embeddings, a far smaller number of clusters is necessary and even advantageous. More detail about the exact training parameters is given in Section 9.3.5. This training protocol will be referred to as *biased* and is presented in Algorithm 2.

Algorithm 2 The biased Masked-Prediction Pre-Training (MPPT) and fine-tuning.

```
Let $I$ be the number of iterations
Let $\mathcal{D}_l$ be the labelled data set
Let $\mathcal{D}_{ul}$ be the unlabelled
Set $i \leftarrow 0$
while $i < I$ do
    Decide which vectors to cluster. MFCC features for $i = 0$ otherwise choose layer of $\mathcal{M}_{i-1}$ based on label purity.
    Cluster $\mathcal{D}_{ul}$ to obtain tokenisation $\mathcal{T}$
    Randomly initialise the model $\mathcal{M}_i$
    Train $\mathcal{M}_i$ on $(\mathcal{D}_{ul}, \mathcal{T})$
    Fine-tune $\mathcal{M}_i$ on $(\mathcal{D}_l)$ for a small number of updates (Biasing step)
end while
Set $\mathcal{M}_I$ to be $\mathcal{M}_{I-1}$ with replaced output layer (see Figure 9.1b)
(Fully) fine-tune $\mathcal{M}_I$ on $\mathcal{D}_l$
return $\mathcal{M}_I$
```

9.3 Experiments for Biased and Unbiased MPPT/Fine-tuning

9.3.1 Data

The proposed methods were evaluated on the LibriSpeech data set (Panayotov et al., 2015). It contains 960 hours of data from audiobooks. A random 100-hour subset serves as the supervised data set, $\mathcal{D}_l$, that is used for biasing and fine-tuning. The remaining 860-hour data set serves as the unsupervised data set, $\mathcal{D}_{ul}$. For the masked-prediction pre-training, the complete 960 hours of data are used for training. The utterances in LibriSpeech were segmented to be of length ≤ 10 s. The official dev-other data set is used for all the label-statistics calculations (see Section 9.1.3). The labels were generated by aligning the dev-other data set using a bi-grapheme system that used a set of 870 clustered bi-graphemes. For evaluation, both the test-clean and the test-other data sets are used. Detailed information about the LibriSpeech data set is provided in Section A.2.

9.3.2 Model

The encoder model structures used throughout this chapter have several differences from the ones used in (Hsu et al., 2021a) (see Sections 4.8.2.2 and 4.8.2.3 for details). The convolutional encoder with an output frequency of 50 Hz, which processes the raw-waveform input, is replaced with a conventional 80dim FBANK analysis operating at 100 Hz with a 25 ms window size. The size of the convolutional kernel of the 1D-Convolution layer used for relative positional encodings is reduced from 127 (at 50 Hz, which is equivalent to 2.54 seconds) down to 31 (at 100 Hz, which is equivalent to 0.31 seconds) for increased efficiency. This layer is followed by stacking four frames which reduces the frequency of the encoder outputs down to 25 Hz. Furthermore, the Transformer layers (Vaswani et al., 2017) are replaced by the much more efficient Emformer layers (Shi et al., 2021) (see Section 2.1.8). The encoder has 20 Emformer layers with an embedding dimension of 512, a feed-forward dimension of 2048, and 8 attention heads. The Emformer layers use a right-context of 1 and a main segment size of 160. The projection layer (see Figure 9.1a) has a size of 256.

9.3.3 Notation

The models pre-trained with masked-prediction pre-training are given a model ID X_Z^Y. X relates to the vectors that are clustered to obtain the label-sequence for training, *e.g.* M for 39dim-MFCC features (these are 13dim-MFCCs plus deltas and delta-deltas), where Y relates to how many updates the model was trained *e.g.* 100k for 100k updates and Z relates to the number of clusters used, *e.g.* 100. Hence, M_{100}^{100k} is a model trained for 100k updates for which the training labels are obtained by clustering 39dim-MFCC features using K=100.

9.3.4 Baseline Pipeline

The baseline pipeline is based on the HuBERT-Base (Hsu et al., 2021a) pipeline and is entirely unsupervised. The first iteration of masked-prediction pre-training uses a label sequence obtained by clustering 39dim-MFCC input features (these are 13dim-MFCCs plus deltas and delta-deltas) features using K=100. These features are computed at 100 Hz, and the label sequence is sub-sampled by a factor of four to align with the 25 Hz output frequency of the encoder model. The encoder models that are trained for 100k and 250k updates have the IDs M_{100}^{100k} and M_{100}^{250k}, respectively. These encoder models will also be referred to as iteration-1 models. MFCC features are used instead of FBANK features because of their better match with the feature independence assumption of the squared distance metric of the K-Means clustering algorithm. The second iteration of training uses a label sequence obtained by clustering the embeddings of one of the layers of M_{100}^{250k} using K=500. This layer was chosen based on the cluster purity which was maximised for layer 10. This is similar to HuBERT-Base where for a 12 layer model, layer 6 maximises the label-purity *i.e.* the layer in the centre of the model seems to be optimal. The encoder models that were trained for 100k, 250k and 400k updates have the IDs H_{500}^{100k}, H_{500}^{250k} and H_{500}^{400k}, respectively. These models will also be referred to as iteration-2 models.

9.3.5 Biased Pipeline

For the biased pipeline, the starting point was the unbiased iteration-1 model that was trained for 250k updates (*i.e.* M_{100}^{250k}). This model is fine-tuned for 20k updates using a CTC loss on the labelled data. This is the biasing step. These 20k updates come after the linear output layer was trained for 10k updates itself, as is done for the fine-tuning (see Section 9.3.7) as well. Fine-tuning for only 5k updates also gave good results, but training for too long (*e.g.* 80k updates) started to give poorer results, which could be due to overfitting to the supervised subset of the data. It is likely that for a smaller supervised data set (say 10 h) fewer updates should be used in the biasing step. Then the embeddings of the 17-th Emformer-layer were clustered to obtain the target sequence. The 17-th layer was picked as it maximised the label-purity. The more similar the model is to an ASR model, the closer is the ideal layer for the embeddings to the output. For the biased training, both K=500 and K=100 were tested. These pre-trained models are denoted C_{β}^{α}. A model that is pre-trained using the clustering the embeddings of C_{100}^{250k} as its tokenisation is denoted S_{β}^{α}.

9.3.6 Training Parameters

For the masked-prediction pre-training of any iteration (biased or unbiased), around 50% of the input frames are masked by applying overlapping masks of length 20 frames (equivalent to 200 ms; (Hsu et al., 2021a) used masks length 10 for 50 Hz features *i.e.* also 200 ms). This is done by choosing 4% of the frames as starting frames for the masks of length 20. The classification loss is computed only for the masked frames. The models were trained on 32 GPUs with a batch size of 87.5 seconds per GPU (for 250k updates, this is equivalent to 169 epochs). The learning rate scheduler ramped up linearly for the first 8% of updates and then decayed linearly down to zero. The peak learning rate is 5e-4.

ID	Embeddings		K	T-steps	M-acc
	Model	Layer			
M_{100}^{100k}	39D-MFCC		100	100k	48.4
M_{100}^{250k}	39D-MFCC		100	250k	49.5
H_{500}^{100k}	M_{100}^{250k}	10	500	100k	58.3
H_{500}^{250k}	M_{100}^{250k}	10	500	250k	59.1
H_{500}^{400k}	M_{100}^{250k}	10	500	400k	61.0
C_{500}^{100k}	M_{100}^{250k} + ft.	17	500	100k	67.7
C_{500}^{250k}	M_{100}^{250k} + ft.	17	500	250k	70.1
C_{500}^{400k}	M_{100}^{250k} + ft.	17	500	400k	71.5
C_{100}^{100k}	M_{100}^{250k} + ft.	17	100	100k	77.0
C_{100}^{250k}	M_{100}^{250k} + ft.	17	100	250k	79.1

Table 9.1 Masked-prediction pre-training. ID is the ID of the model that was trained. Embeddings are the vectors clustered into K clusters to obtain the training sequence. M-acc is the model's accuracy on the masked frames on the validation set. T-steps is the number of training updates. '+ft.' indicates the biasing step.

9.3.7 Supervised Baselines and Fine-tuning

The pre-trained models were fine-tuned using the CTC loss (Graves et al., 2006). The output units were 5000 sentence pieces (Kudo and Richardson, 2018) with Byte Pair Encoding (BPE) (Sennrich et al., 2016) as the segmentation algorithm. For fine-tuning, the projection and output layer of the pre-trained models are replaced with an output layer (see Figure 9.1) mapping to the 5001 output units (including the blank unit that is required in CTC-systems; see Section 3.3.6). The pre-trained encoder is frozen for the first 10k updates, and the entire model was trained for 100k updates (10k+90k). The peak learning rate was 2.5e-5, which is 40x smaller than for the purely supervised models (1e-3). The purely supervised models have the same model structure as the pre-trained models. All models used SpecAugment (Park et al., 2019) without time-warping. For decoding, the official LibriSpeech 4-gram language model was used.

9.3.8 Baseline and unbiased experiments

The masked prediction accuracies are given in Table 9.1. The label-statistics are given in Table 9.2 and WERs are given in Table 9.3. The WERs quoted in the text will refer to the LibriSpeech test-other except when explicitly mentioned.

Embeddings		K	cluster-	label-
Model	Layer		purity	purity
39D-MFCC		100	0.074	0.064
M_{100}^{250k}	10	500	0.079	0.162
M_{100}^{250k}	10	100	0.152	0.114
M_{100}^{250k} + 20k ft.	17	500	0.299	0.197
M_{100}^{250k} + 20k ft.	17	100	0.330	0.194
M_{100}^{250k} + 5k ft.	17	100	0.330	0.195
M_{100}^{250k} + 80k ft.	17	100	0.301	0.174

Table 9.2 Label purity is the label prediction accuracy if the cluster is known. Cluster purity is the cluster prediction accuracy if the label is known. Label purity goes up for higher K. Cluster purity goes down for higher K. Labels are a set of 870 clustered bi-graphemes. '+ft.' indicates the biasing step for 5k, 20k or 80k update steps.

The WERs of the baseline 100-hour and 960-hour purely-supervised models were 17.76% and 7.37%, respectively. The 960-hour results (7.37%) is the 'oracle' result in Equation (9.3), *i.e.* $\text{WER}_{\text{super}} = 17.76\%$ and $\text{WER}_{\text{oracle}} = 7.37\%$.

From Tables 9.1 and 9.3 it can be seen that training for longer keeps improving the model's accuracy. For the M_{100}^{100k} and M_{100}^{100k} models the WERs after fine-tuning these unbiased iteration-1 models were 14.01% and 12.68%, respectively. For the H_{500}^{α} models there is still a significant improvement in the model's accuracy when going from 250k updates to 400k updates. This improvement is in fact larger than the improvement from 100k updates to 250k. The corresponding WERs after fine-tuning these unbiased models were 13.00%, 10.72% and 10.23% for H_{500}^{100k}, H_{500}^{200k} and H_{500}^{400k}, respectively.

For comparison Hsu et al. (2021a) reports $\text{WER}_{\text{oracle}} = 5.8\%$ and a semi-supervised result of 8.1% WER for HuBERT-Base, which uses a less efficient encoder architecture.

These results show that training for at least 250k updates is absolutely necessary for these models as the relative improvement from the 150k extra update steps were 9.5% and 17.5% respectively for the M_{100}^{α} and H_{500}^{α} models. It also shows that the second iteration of training is very important.

The clusters derived from layer 10 of M_{100}^{250k} give better predictions of the bi-grapheme labels than the clusters from the MFCC features (label-purity of 0.114 vs 0.064) for K=100.

Looking at Table 9.1, it can be seen that even with the increase in the number of clusters from 100 to 500, the masked-prediction accuracy is higher for H_{500}^{250k} than for M_{100}^{250k} (59.1% vs 49.5%). This is likely due to the tokenisation being more consistent.

Data	Init.	test-clean	test-other
100h	-	8.68	17.76
100h	M_{100}^{100k}	6.98	14.01
100h	H_{500}^{100k}	6.52	13.00
100h	C_{500}^{100k}	5.28	10.39
100h	C_{100}^{100k}	5.06	9.91
100h	S_{100}^{100k}	4.64	8.93
100h	M_{100}^{250k}	6.26	12.68
100h	H_{500}^{250k}	5.27	10.72
100h	C_{500}^{250k}	4.67	9.60
100h	C_{100}^{250k}	4.48	9.06
100h	S_{100}^{250k}	4.50	8.75
100h	H_{500}^{400k}	5.02	10.23
100h	C_{100}^{400k}	4.40	8.94
960h	-	3.26	7.37

Table 9.3 WER results on LibriSpeech test sets after CTC fine-tuning. Data is hours of fine-tuning data. ‘Init.’ is the model used for initialisation.

9.3.9 Biased Training Results

To recall, before clustering, the iteration-1 model, M_{100}^{250k}, was fine-tuned for 20k update steps and the embeddings of the 17-th layer clustered. For K=500, the purity statistics improve significantly (see Table 9.2) over the unbiased models. The cluster-purity increased from 0.079 to 0.299, and the label-purity from 0.162 to 0.194. The strong increase in cluster-purity confirms the idea that without supervision, many cluster tokens represent the same phonetic unit but in different acoustic contexts (speaker, channel, noise). Part of the challenge of acoustic modelling is to be invariant to these changes in acoustic context. Thus, a higher cluster-purity is desired. Another effect of the biasing is that K=100 seems to be large enough to model the bi-grapheme labels. For the unbiased embeddings, the label-purity dropped from 0.162 down to 0.114 when changing K=500 down to K=100. For the biased embeddings, only a small drop from 0.197 to 0.194 was observed. This observation motivated the decision to try K=100 for the biased second iteration of training and not only K=500 as commonly used (Chen et al., 2022; Hsu et al., 2021a; Wang et al., 2022). Table 9.2 also shows that for a larger number of fine-tuning updates (80k), the embeddings become worse as measured by the label-purity and the cluster-purity metrics. This could be due to overfitting to the supervised data set.

From Table 9.1, it can be observed that the biased labels are much cleaner and more predictable. The masked prediction accuracy, after 250k training steps, increases from 59.1% to 70.1% (H_{500}^{250k} vs C_{500}^{250k}). Using K=100 has a masked prediction accuracy of 79.1% (*i.e.* C_{100}^{250k}).

For 250k updates of training, the biased model (initialised from C_{500}^{250k}) outperforms the unbiased model (initialised from H_{500}^{250k}) by 10.4% (10.72% WER vs. 9.60% WER in Table 9.3). Using K=100 further reduces the WER to 9.06% (overall a 15.5% rel. WER reduction). Therefore the biased training is able to recover 83.7% of the reduction in WER that could be obtained by using 860 hours of labelled data in comparison to the 67.8% WER recovery achieved by the unbiased pre-training (see Equation (9.3) for WER recovery).

The improvements due to the biasing are even stronger when training for only 100k updates. The relative WER reduction over the unbiased baseline is 20.1% for K=500 and 23.8% for K=100. This demonstrates that biased self-supervised learning improves not only the final performance but also the convergence speed. The main reason for this is likely the improved consistency of the label sequences and the thus increase in the predictability of the cluster-tokens, which can be seen from the increase in masked-prediction accuracy in Table 9.1. Label noise in supervised learning, which can be seen as analogous to less consistency in the tokenisation obtained from clustering, hinders convergence as shown by Zhang et al. (2017).

For a biased iteration-3 model where the label sequence is obtained by clustering the embeddings of C_{100}^{250k}, the convergence speed increases even more (this model is called S_{100}^{α} in Table 9.3). Even though the WER for 250k updates is only 3.4% lower (8.75% vs 9.06%) than for the biased iteration-2 model (both with K=100), for 100k updates, the WER is 9.9% lower (8.93% vs 9.91%). This again reinforces the connection between how clean the label-tokens are and the convergence speed. The biased 250k iteration-3 model, S_{100}^{250k}, recovers 86.7% of the reduction in WER obtained from 860 hours of labelled data (17.76% $\rightarrow$ 7.37% vs 17.76% $\rightarrow$ 8.75%).

As another point of comparison, noisy student-teacher training (Park et al., 2020) (a form of self-training) reports a WER recovery of 64.8%. To achieve such high WER recovery using self-training, four iterations of self-training were performed, and the official LibriSpeech 4-gram language model was used not only for evaluation but also for the decoding step for each iteration of self-training. For each iteration the acoustic model was trained on 32 Google Cloud TPU chips for ten days.

9.4 Low-footprint streaming models

Streaming ASR has two major operating points. Server-side streaming ASR aims to recognise the speech in real-time, but does not have memory constraints which leads to the ability to have

Data	Initial model	test-clean	test-other
100h	-	8.68	17.76
100h	M_{100}^{100k}	6.98	14.01
100h	H_{500}^{100k}	6.52	13.00
100h	C_{100}^{100k}	5.06	9.91
100h	S_{100}^{100k}	4.64	8.93
100h	above -Conv1D-Layer + CE	4.95	10.30
100h	above with K=2000	4.81	9.93
960h	-	3.26	7.37

Table 9.4 Results on LibriSpeech test sets after CTC fine-tuning. Data is hours of fine-tuning data. Initial model is the model used for initialisation. '-Conv1D-Layer' means that the model's 1D-convolutional layer was removed. 'CE' refers to the training style outlined in Section 9.4. Results for 'CE'-style pre-training are compared to a selection of results from Table 9.3.

an unlimited left-context (view into the past). On-device streaming ASR has the requirement for limited memory, which requires the input context to be smaller. Masked-prediction pre-training is challenging for streaming models that have small input contexts. The reason for this is that too much of the input is masked in order to make accurate predictions. Making the task prohibitively difficult leads to poor representations being learned. Furthermore, removing the 1D-Convolutional layer further improves efficiency but would make the masked-prediction pre-training task even harder as it would reduce the quality of the positional encoding. This part of the chapter aims to solve this issue by computing the classification loss on both the masked and the unmasked frames. By treating the clusters obtained from clustering the embeddings of a second-iteration biased model C_{100}^{250k} as acoustic units, the loss obtained from the unmasked frames can be treated as Cross Entropy (CE) training for DNN-HMM hybrid models (Bourlard and Morgan, 1994; Dahl et al., 2012) (see Section 3.9.2). In practice, the input was masked as before, but the loss-weight on the masked frames and the loss-weight on the unmasked frames are both 0.5. Most of the learning comes from the unmasked frames, and the masking serves as a form of regularisation.

9.4.1 Experiments with low-footprint streaming models

The experimental setup for this section is very close to the setup from Section 9.3. For the new results in Table 9.4, the just explained training style is used. The results in Table 9.5 use a low-footprint streaming architecture for the audio encoder.

Embeddings		Data sets	
Model	Layer	test-clean	test-other
(Supervised)		12.45	24.07
H_{500}^{250k}	15	8.14	16.41
C_{100}^{250k}	18	6.13	13.34

Table 9.5 Results on LibriSpeech test sets after CTC fine-tuning. The trained models are low-footprint streaming models with an algorithmic latency introduced by the encoder of 120 ms. The models are trained using the training style of Section 9.4. 'Embeddings' refers to the embeddings used for the pre-training tokenisation.

First, the CE-style pre-training is demonstrated for the non-streaming models but with the 1D-Convolutional layer removed. The labels are the same as for the biased iteration-3 model S_{100}^{α}. In the first experiment, this style of training was used while at the same time removing the 1D-Convolutional layer. This increases the WER from 8.93% to 10.30%, as shown in Table 9.4. Using a larger token set of K=2000 reduces the WER down to 9.93%. Here, a larger token set helps because another interpretation of this training method is that the model is distilled from C_{100}^{250k}. A larger token set gives more information about C_{100}^{250k}.

For the low-footprint streaming models, the Emformer layers have a main segment size of 4 (see Section 2.1.8), equivalent to 160 ms, and the 1D-Convolution layer is removed. They still have a right-context of 1 equivalent to 40 ms. This yields an average algorithmic latency induced by the encoder of 120 ms. For the tokenisations, K=2000 was used for the reason mentioned above.

For this low-footprint streaming model, the 100-hour supervised WERs were 12.45% and 24.07% WER on test-clean and test-other, respectively. The non-streaming models that were used in Section 9.3 achieved WERs of 8.68% and 17.76%. The standard masked-prediction pre-training gave poorer WER results than not using it. Using the proposed training style based on CE-training for HMM-Deep Neural Network (DNN) models, the WERs were reduced to 6.13% and 13.34% for the same low-footprint streaming model. This is a relative reduction in WERs of 50.8% and 44.6%. Training in the CE style, but using labels derived from clustering the embeddings of the 15-th layer of H_{500}^{250k} leads to WERs of 8.14% and 16.41%. Therefore, the CE-style pre-training made pre-training these low-footprint streaming models possible, while the biased self-supervised learning gave significant improvement over the unbiased version.

Overall it can be said that the CE-style pre-training based on either a biased or an unbiased tokenisation can pre-train models of any model architecture.

9.5 HMM-based tokenization

The tokenisation that was used in the previous sections to obtain the label sequence operated on the frame-level using K-Means. Therefore the sequence level structure of the sequences is not taken into account. The labels derived using K-Means can have outliers which causes the label-sequence to flip backwards and forwards between adjacent frames. The tokenisations that are derived using K-Means have been used for other tasks (Lakhotia et al., 2021; Lee et al., 2022b; Wu et al., 2022). Often de-duplicated labels are used, which means that consecutive occurrences of the same discrete label are merged into one. In Wu et al. (2022) an end-to-end trainable ASR model is pre-trained by treating de-duplicated cluster tokens as graphemes. Lee et al. (2022b) performs speech-to-speech translation where de-duplicated cluster tokens of the target language are used as an intermediate representation. Lakhotia et al. (2021) uses cluster tokens to directly synthesise speech.

To try to improve on the labels derived using K-Means, this section proposes the use of a HMM for clustering, which is inspired by the use of HMMs in ASR. The HMM could provide a temporally smoothed version of the K-Means tokenisation. This should improve the quality of the de-duplicated labels and could also help the training of HuBERT models. The reduced label noise can speed-up training as shown already in this chapter. Two HMM structures will be examined. One is a fully-connected HMM, where the transition matrix does not have any zero entries. The second HMM structure will be a minimum-duration HMM where if the HMM enters a state, it will have to stay in it for at least three steps. These HMMs were built using the hmmlearn toolkit[1].

9.5.1 Fully-Connected HMM

The K-Means algorithm is analogous to a Gaussian Mixture Model (GMM), where the GMM has identity matrices as the covariance matrices and the prior distribution is uniform. The HMM that would be most similar to the GMM would have single Gaussian emission probabilities and an identity covariance matrix. Here these restrictions are slightly relaxed, and a diagonal covariance matrix is used. It is important to note that there are differences in the way in which HMMs are used here and how they are typically applied in HMM-based acoustic models for ASR. Here, there are no multiple constituent HMMs that are strung together into a large left-to-right HMM (as in Figure 3.3), but only one single completely unsupervised HMM. In the experiments, the transition matrix was uniformly initialised, the covariance matrices were initialised to be the identity matrix, and the mean-vectors were initialised by first training a K-Means model and using the centroids as mean-vectors. A variance floor was used to prevent

[1]Available at https://hmmlearn.readthedocs.io/en/latest/

covariance collapse. To provide the tokenisation for an input sequence the Viterbi algorithm (see Section 3.5.1) was used.

9.5.2 Minimum-duration HMM

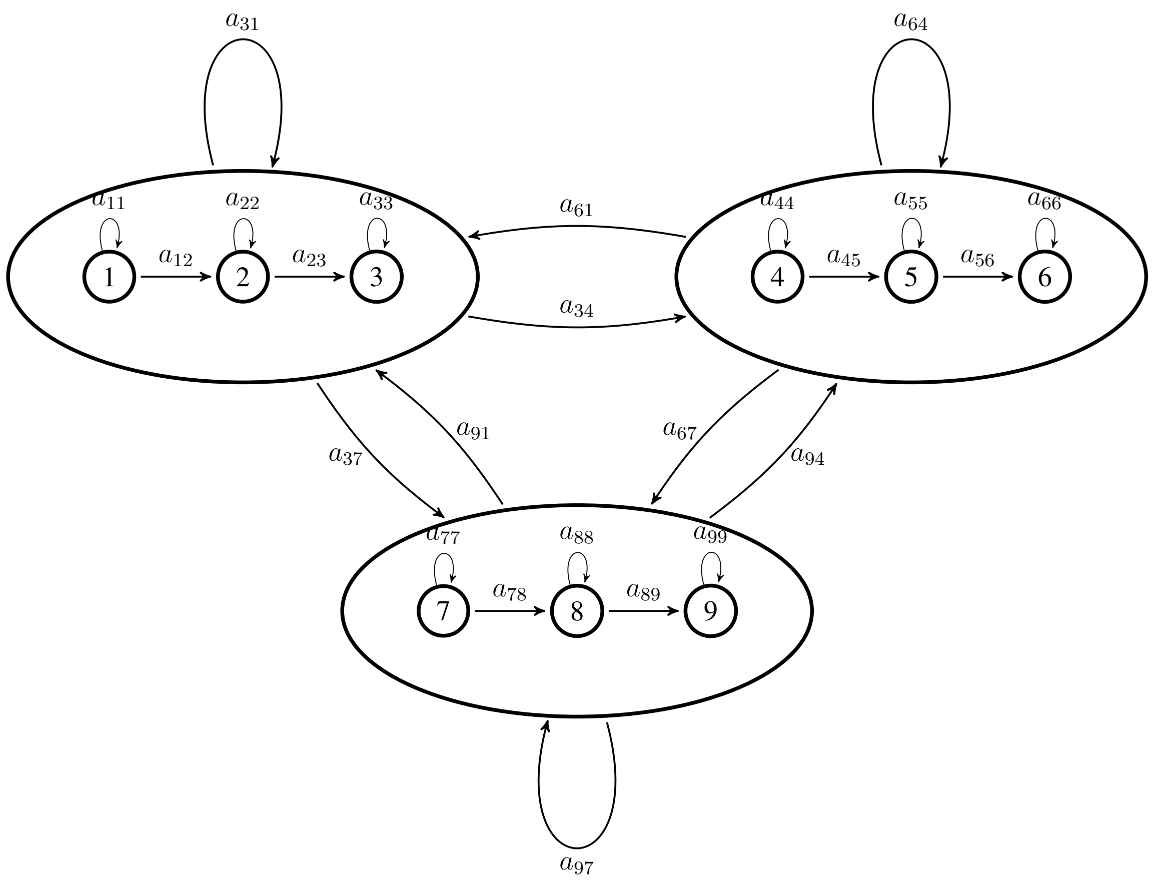


Figure 9.2 Minimum duration HMM for K=3

In an HMM, the transition matrix can be structured in order to bias the HMM towards certain behaviour. The goal is to have the HMM stay in the same state for longer. Here, there are $3K$ states used, which are grouped into groups of three (*i.e.* into K groups). If the HMM enters state i (where i mod $3 == 0$), it first has to go through states $i + 1$ and $i + 2$ before it can enter any new state j (where j mod $3 == 0$). Each state also allows for self-transitions. By treating each of the K groups of states as one discrete label, the duration of each label is at least three states. Figure 9.2 shows the structure of such HMM for $K = 3$ with all its non-zero transition probabilities. Effectively, this HMM structure uses K different constituent HMMs (see Figure 9.3) that are overall fully-connected. The mean vectors and covariance matrices are initialised to be the same within each group, and are initialised as for the fully-connected

HMM (see Section 9.5.1). The transition probabilities are initialised to be uniform given the constraints. Figure 9.4 shows such an initialisation. If the current state is the first or second state of one of the K groups (*e.g.* state 0 or 1 in Figure 9.4), the transition probability to the next state is 0.5, and the self-transition probability is also 0.5. If the current state is the third state of one of the K groups (*e.g.* state 2 in Figure 9.4), the transition probability to any of the K starting states is $1/(K+1)$ and the self-transition probability is also $1/(K+1)$. The overall structure of this minimum-duration HMM is inspired by the phone-based HMMs that are used for ASR (see Section 3.3.1).

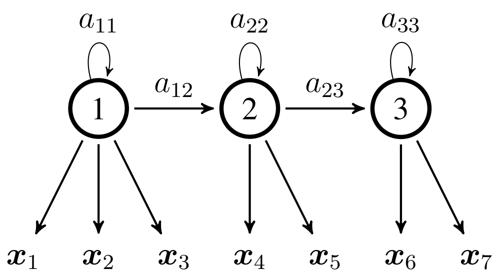


Figure 9.3 Individual constituent HMM of the larger minimum duration HMM

9.6 Experiments for HMM-based tokenisation

The experiments follow the setup of the unbiased experiments in Section 9.3. The input vectors that are used for K-Means clustering and for the HMM-based tokenisation are 39dim-MFCC features (these are 13dim-MFCCs plus deltas and delta-deltas) and $K = 100$ is used (*i.e.* the fully connected HMM has 100 states and the minimum-duration HMM has 300 states with 100 groups). The results for K-Means in this section are identical to the results of M_{100}^{100k} in Section 9.3.

9.6.1 EM-Training of HMMs

After EM-training, the transition matrix of the fully-connected HMM is shown in Figure 9.5. It has a strong diagonal structure which shows that it has many high-probability self-transitions. There is some structure in the transition matrix such that some particular states often follow another particular state. These could be certain phones that often follow each other and thus exhibit these high-probability transitions *i.e.* this structure could be akin to a phone-level language model. Also, the number of states is larger than the number of phones. Hence, this state-level language model could also represent some context-dependency.

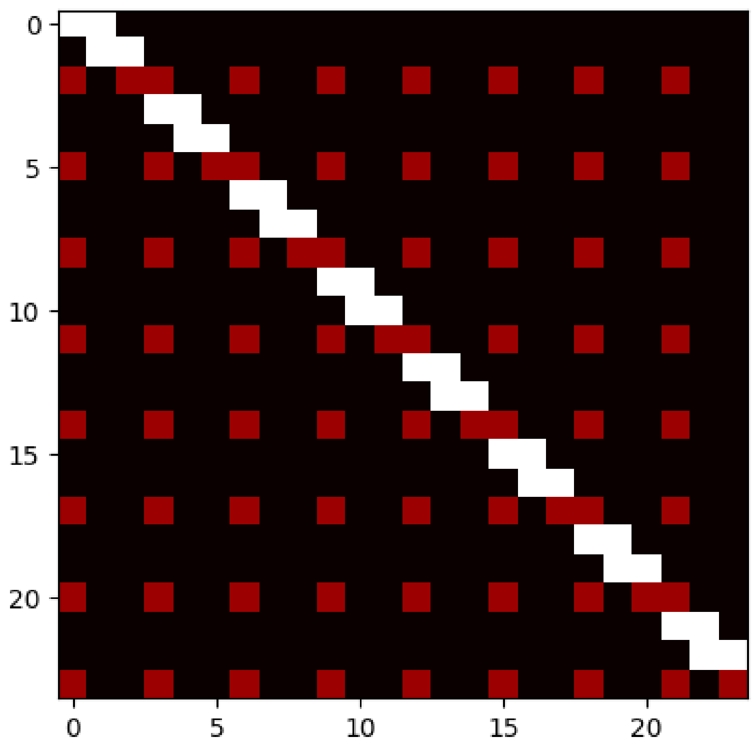


Figure 9.4 Hidden Matrix initialisation of minimum-duration HMM. A 24x24 section is shown. White represents a probability of 0.5. Dark-red represents a probability of $\frac{1}{K+1}$.

After EM-training, the transition matrix of the minimum-duration HMM is shown in Figure 9.6. Within many groups, the third-state doesn't have a significant self-transition probability whilst the first two do. This could indicate that the first two states model the acoustics of the acoustic unit that is being modelled and the third state is a transitory state. In the second group that is shown in the Figure, a transition from the third to the first constituent state is observed. Similar to the transition matrix of the fully-connected HMM in Figure 9.5 the transition probabilities between groups are not uniform and certain groups follow each other more often than others.

9.6.2 Pre-training using the HMM-based tokenisation

Table 9.6 shows the statistics associated with the labels *i.e.* the cluster and label purities. It also shows the masked-prediction accuracy and the unmasked-prediction accuracy (UM-Acc in Table 9.6). All models trained with masked-prediction pre-training (both in this section as well

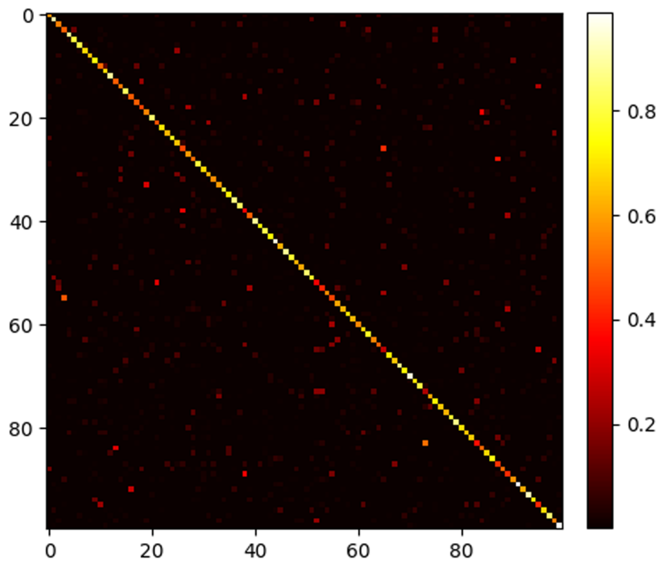


Figure 9.5 Transition matrix of the fully-connected HMM (100 states) after EM-training.

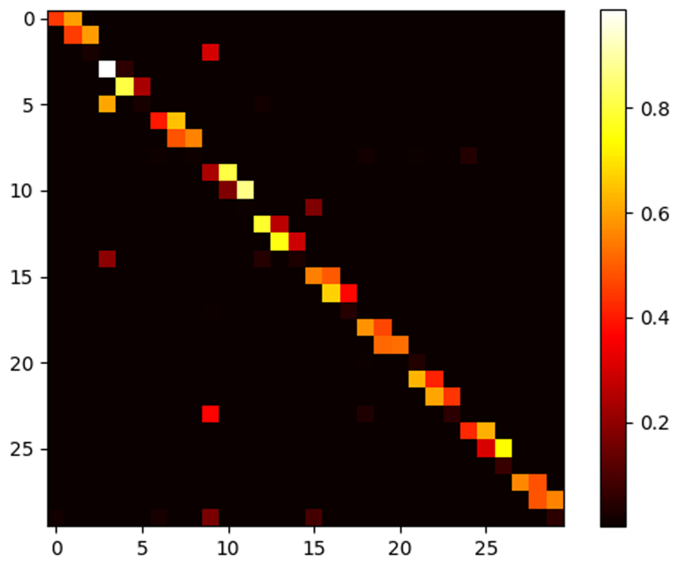


Figure 9.6 Hidden Matrix of minimum duration HMM after EM-training. Only a 24x24 section is shown out of the 300x300 transition matrix.

as those in Section 9.3) are able to correctly predict a large proportion of the discrete labels for unmasked frames even though they were never trained on this task. In a similar way to

Vectors	tokenisation algorithm	K	cluster-purity	label-purity	M-Acc	UM-Acc
39dim-MFCC	K-Means	100	0.074	0.064	48.4	67.8
39dim-MFCC	FC-HMM	100	0.096	0.074	50.4	75.4
39dim-MFCC	sparse HMM	100	0.104	0.081	51.3	78.1

Table 9.6 Label purity is the label prediction accuracy if the cluster is known. Cluster purity is the cluster prediction accuracy if the label is known. Label purity goes up for higher K. Cluster purity goes down for higher K. Labels are a set of 870 clustered bi-graphemes. M-Acc is the masked prediction accuracy and UM-Acc is the accuracy on unmasked frames after training for 100k updates. The sparse HMM is the minimum-duration HMM.

masked-prediction accuracy, an increase in the unmasked-prediction accuracy also indicates an improvement in the consistency of the tokens.

The tokenisation provided by the fully-connected HMM is better correlated with the ground truth labels in comparison to the K-Means algorithm, and the minimum-duration HMM outperforms both. The improvement cluster-purity improves by 29.7% and 40.5% relative over K-Means, and the label-purity improves by 15.6% and 26.5% relative over K-Means.

The increase in masked- and unmasked-prediction accuracy also shows that the labels provided by this improved tokenisation method are much more consistent, which could be due to the smoothing effect of the HMM. The un-masked prediction accuracy improves much more than the masked-prediction accuracy. It can be assumed that a large proportion of the unmasked-prediction errors are due to outlier frames, given that actually all the information needed to make the prediction is available. Therefore, these results directly show the temporal smoothing effect of the HMM-based tokenisation methods that lead to fewer outliers. The tokenisation that uses the minimum-duration HMM reduces the unmasked-prediction errors by 32%.

The HMM based clustering also improves the final WER after fine-tuning as shown in Table 9.7. The K-Means tokenisation achieves a WER of 14.01% on test-other, which is a WER-recovery of 36.1% (same result as in Section 9.3.8). The tokenisation that uses the fully-connected HMM results in a WER of 13.5%, which is a WER-recovery of 41.0%. The tokenisation that uses the minimum-duration HMM results in a WER of 13.17%, which is a WER-recovery of 44.2% and a 6.0% relative reduction in WER over the K-Means tokenisation. Overall, the HMM-based tokenisation methods improve over the K-Means tokenisation based on every single tested metric as shown in Tables 9.6 and 9.7.

Tokenisation method	test-clean	test-other
(100h-supervised)	8.68	17.76
K-Means	6.98	14.01
FC-HMM	6.77	13.50
sparse-HMM	6.58	13.17
(960h-supervised)	3.26	7.37

Table 9.7 WER results on LibriSpeech test sets after CTC fine-tuning on 100h of data after 100k updates of pre-training training. Different tokenisation methods that are used to tokenise the 39dim-MFCC features are compared. The K-Means result is identical to the result of M_{100}^{100k} in Table 9.3.

9.7 Summary

This chapter focused on self-supervised learning via masked-prediction pre-training. In particular, it focused on methods to improve the input tokenisation that is used to derive the training targets. To this end, two methods were proposed. The first method *biased self-supervised training* improves the tokenisation by introducing the small amount of supervised data that is usually reserved for fine-tuning into the tokenisation process. The second method uses HMMs instead of K-Means for tokenisation.

In Section 9.2, biased self-supervised training was proposed. Instead of clustering the embeddings of a model trained using unsupervised training, it clusters the embeddings of a model that was fine-tuned for a small number of updates. The fine-tuning is performed using the small amount of supervised data that is available in any semi-supervised learning scenario. This fine-tuning ensures that the self-supervised learning task is specialised towards the task for which the model is supposed to be used. Experiments on the LibriSpeech corpus, where a 100 h subset was picked as the supervised data set and the remaining 860 h were treated as the unsupervised data set, showed that biased self-supervised training yields not only improved ASR performance but also has faster convergence speed. After 100k training updates, the biased models achieved a WER of 9.91% on test-other, which is a 24% reduction in WER over the unbiased baseline and a 44% reduction in WER over the purely supervised baseline. After 250k training updates, the biased models achieved a WER of 9.06% on test-other, which is a 15% reduction in WER over the unbiased baseline and a 49% reduction in WER over the purely supervised baseline. The biased model that was trained for 100k updates also outperformed the unbiased model that was trained for 400k updates (9.91% vs 10.23%).

In Section 9.5, it was proposed to replace the K-Means clustering algorithm that was previously used to tokenise the input with a HMM. The HMM is trained in an entirely unsupervised fashion using the EM-algorithm. The tokenisation of the input is performed using the Viterbi algorithm. The result is a tokenisation algorithm that takes the sequential nature of the data into account and can temporally smooth the tokenisation. Both a fully-connected HMM (a HMM with no zeros in the transition matrix) and a minimum-duration HMM (a HMM with groups of states that are to be traversed left-to-right in at least three steps) were examined. The tokenisation was examined using a variety of metrics and improved every one of them. The minimum-duration HMM performed the best and it reduced the WER over the K-Means algorithm by 6% (from 14.01% on test-other down to 13.17%).

Furthermore, Section 9.4 showed that masked-prediction pre-training can be changed to be more similar to the frame-level CE-training of HMM acoustic models. This allowed low-footprint streaming ASR-models to be trained. Furthermore, using biased self-supervised training resulted in a 19% WER reduction on test-other over the unbiased baseline and a 45% WER reduction over the purely supervised baseline.

Chapter 10

Conclusions and Future Work

This thesis has addressed two fundamental learning paradigms for Automatic Speech Recognition (ASR) that will be central to the more widespread use of ASR in technology in the future: active learning and semi-supervised learning.

Chapter 2 first establishes the fundamentals of deep learning. Chapter 3 introduced the different components of and approaches to ASR-systems. Chapter 4 described various semi-supervised learning methods for general machine learning and for ASR. Chapter 5 outlined the background around active learning for ASR and also detailed the factors that influence the cost of transcribing ASR data. The remaining chapters focused on the following major contributions and novel approaches that were developed:

1. A technique for semi-supervised learning of speaker embeddings termed cosine-distance virtual adversarial training (CD-VAT) was proposed. The method consists of a regularisation loss that is calculated using the unsupervised data set. The loss ensures that the model is smooth with respect to the input on the data manifold. CD-VAT is evaluated and shown to be effective for speaker verification.

2. A technique for active learning in ASR based on Bayesian principles. The particular implementation is termed *NBest-BALD*. It uses the mutual information between the prediction for the utterance and the model parameters as the selection criterion.

3. A technique was proposed that matches the distribution over the utterance lengths for the untranscribed utterance pool and the utterance pool selected by the active learning algorithm. This method alleviates the issue that active learning methods for ASR tend to select shorter utterances.

4. An approach for the use of a realistic cost function for labelling utterances and methods for selecting variable-length groups of utterances that occur in sequence in a conversation or a meeting was proposed.

5. A self-supervised learning technique that combines supervised and unsupervised data was proposed. It is analysed how this method leads to not only better performance, but also faster training.

6. A new method to derive the labels for self-supervised learning based on Hidden Markov Models (HMMs).

These contributions are summarised in detail in the next section. Taking into consideration all of these key contributions, the concluding section puts forth suggestions for possible extensions and exciting opportunities that can be pursued in the future.

10.1 Summary and Conclusions

Chapter 6 began by investigating consistency-regularisation in the form of virtual adversarial training (VAT) for the semi-supervised training of HMM-Deep Neural Network (DNN) acoustic models. It was shown that for frame-level Cross Entropy (CE) training this was not useful due to inherent label noise. Then the novel algorithm CD-VAT was developed for the semi-supervised training of speaker-discriminative acoustic embeddings without the requirement that the set of speakers is the same for the labelled and the unlabelled data. CD-VAT is a form of consistency-regularisation where the supervised training loss is interpolated with an unsupervised loss. This loss is the CD-VAT-loss, which smoothes the model's embeddings with respect to (w.r.t) the input as measured by the cosine-distance between an embedding with and without adversarial noise. For speaker verification using the VoxCeleb data set, it was shown that CD-VAT recovers 32.5% of the Equal Error Rate (EER) improvement that would be obtained when all speaker labels are available for the unlabelled data. Furthermore, two new metrics for analysis of a speaker embedding model were proposed: (i) a metric for the intra-speaker compactness (ISC); and (ii) a metric for the inter-speaker separability (ISS). These two qualities influence the performance of speaker verification and speaker diarisation systems. The two proposed metrics allow researchers to analyse the effects of changing the speaker embedding model in more detail. For example, in the speaker verification experiments in Chapter 6 it was shown that the improvements due to CD-VAT are mostly due to improving the ISC rather than the ISS.

Chapter 7 introduced Bayesian active learning for ASR and specifically via NBest-BALD as a specific active selection algorithm. NBest-BALD selects utterances based on the mutual information between the prediction for the utterance and the model parameters θ, *i.e.* $\mathcal{I}[\theta, \boldsymbol{w}|\mathcal{D}_l, \mathbf{X}_i]$. Monte-Carlo Dropout is used to approximate sampling from the posterior of the model parameters, and an N-Best list is used to approximate the entropy over the hypothesis space. Experiments on the Switchboard corpus showed that NBest-BALD outperforms random sampling and prior active learning methods that use confidence scores or the NBest-Entropy as the informativeness measure. NBest-BALD increases the absolute Word Error Rate (WER) reduction obtained from selecting more data by up to 14% as compared to random selection. Furthermore, the selected data was shown to be transferable to a different acoustic model structure.

Chapter 8 focused on two issues of current active learning methods. The first issue is the fact that active learning often does not perform well for the acquisition of the first batch (if the batch is small). The second issue is the fact that previous active learning publications regarded utterances as completely independent and did not take into account the reduced cost of transcribing utterances that occur in a sequence into account. The first issue was addressed via a novel method for encouraging representativeness in active data selection was developed. The method first builds a histogram over the lengths of the utterances. Then, to select an utterance, a word length is sampled from the histogram and the utterance with the highest informativeness within the corresponding histogram bin is chosen. This ensures that the selected data set has a similar distribution of utterance lengths to the overall data set. Experiments on the Switchboard corpus showed that this method significantly improves the performance of active learning for the first batch. The histogram-based sampling increased the absolute WER reduction obtained from selecting more data by up to 57% as compared to random selection and by up to 50% as compared to an approach using informativeness alone. The second issue was addressed via the introduction of a cost function for transcribing ASR data which takes into account the sequential nature of conversations and meetings. The cost function combines a Real-Time Factor (RTF) for the utterance length (in seconds) with an overhead for each utterance (t_1) and an overhead for a chunk of consecutive utterances (t_2). Further, methods were proposed for selecting data on the utterance-level, as fixed-length chunks of consecutive utterances, as variable-length chunks of consecutive utterances and on the side-level whilst taking into account the newly proposed cost function. The overhead t_2 affects the utterance-level selection methods (which are used by methods in the literature) the most, and this method yielded the worst ASR performance. This showed the importance of using the correct cost function and for it a suitable selection method.

Chapter 9 focused on self-supervised learning via masked-prediction pre-training. In particular, it proposed two methods to improve the input tokenisation that is used to derive the training targets. The first is biased self-supervised training. Instead of clustering the embeddings of a model that is trained using unsupervised training, the embeddings of a model that was finetuned for a small number of updates are clustered. The finetuning is performed on the small amount of supervised data that is available in any semi-supervised learning scenario. This finetuning ensures that the self-supervised learning task is specialised towards the task for which the model is supposed to be used. Experiments on the LibriSpeech corpus showed that biased self-supervised training yields not only improved ASR performance but also had faster convergence speed. Biased self-supervised learning reduced the WER by up to 24% over the unbiased baseline. Overall biased self-supervised achieved up to 87% of the reduction in WER that would be achieved if all the unsupervised data were transcribed. The second proposed method is to replace the K-Means clustering algorithm that was previously used to tokenise the input with a HMM. The HMM is trained in an entirely unsupervised fashion using the EM-algorithm. The tokenisation of the input is performed using the Viterbi algorithm. The result is a tokenisation algorithm that takes the sequential nature of the data into account and can temporally smooth the tokenisation. Both a fully-connected HMM (a HMM with no zeros in the state transition matrix) and a minimum-duration HMM (a HMM with groups of states that are to be traversed left-to-right in at least three steps) were examined. The tokenisation was examined using a variety of metrics and improved every one of them. The HMM-based tokenisation reduced the WER by up to 6% as compared to the tokenisation that uses K-Means. Furthermore, Chapter 9 showed that masked-prediction pre-training can be changed to be more similar to the frame-level CE-training of HMM acoustic models, which allows a larger variety of model architectures to be pre-trained. These include low-footprint streaming ASR-models, for which this form of training yielded a reduction in WER of up to 35% when using the unbiased tokenisation and of up to 51% when using the proposed biased tokenisation.

10.2 Future Work

Since the development of CD-VAT, self-supervised learning algorithms have been used for semi-supervised learning for speaker verification with impressive results (Chen et al., 2022; Yang et al., 2021). During the fine-tuning stage that always follows the self-supervised representation learning, CD-VAT can still be used and should be explored.

Self-supervised learning algorithms that are inspired by SimCLR (Chen et al., 2020) (see Section 4.8) use the similarity of the embeddings of the same input transformed by two data augmentations as the objective. These methods have been investigated for ASR (Baevski et al., 2022; Jiang et al., 2021; Khorram et al., 2022). The data augmentations were types of input noise, speed perturbation and input masking. The adversarial perturbations of VAT would provide another type of noise to be used for this type of self-supervised learning.

Future work regarding NBest-BALD should be focused on improving its computational efficiency. This could be focused on reducing the number of Monte-Carlo samples that are used. Furthermore, NBest-BALD should be combined with the histogram-based sampling approach that was proposed in Chapter 8, which could replace the current use of the similarity scores based on term frequency inverse document frequency (TF-IDF) vectors.

Future work regarding biased self-supervised learning should attempt to bias the algorithm towards tasks other than ASR and especially multi-task biasing. Furthermore, the gains in performance and training speed that are provided by biased self-supervised learning does not have to come at the expense of having universally useful representations. Multi-task biasing can still be used to focus the learning of the neural network towards relevant variation in the data and to remove useless information as the information flows through the layers of the network.

Furthermore, research regarding text-less natural language processing approaches, speech synthesis or speech-to-speech translation approaches that use unbiased tokenisation via K-Means has recently garnered increased popularity (Lakhotia et al., 2021; Lee et al., 2022a,b; Polyak et al., 2021). These approaches could benefit significantly from both biasing the embedding model and also the HMM-based tokenisation. Especially the approaches that rely on deduplicated tokens could benefit from the HMM-based tokenisation. Deduplication of the token sequence means that if the same token occurs multiple times consecutively, the deduplicated sequence will contain only one (*e.g.* `112123233` becomes `1212323`). Ideally, the temporal smoothing effect provided by the ASR system would provide the token sequence `111222333` which would be deduplicated to be `123`.

Appendix A

Datasets

This appendix provides detailed information about the major data sets used in this thesis, including Voxceleb, Switchboard and LibriSpeech. Voxceleb was used in Chapter 6. Switchboard was used in Chapters 7 and 8. Librispeech was used in Chapter 9.

A.1 Voxceleb

The Voxceleb family of data sets (Nagrani et al., 2019) consist of utterances that were obtained from YouTube videos and automatically labelled using a visual speaker recognition pipeline. Two Voxceleb data sets exist; VoxCeleb1 (dev+test) (Nagrani et al., 2017) and VoxCeleb2 (dev+test) (Chung et al., 2018). The data sets contain 352 hours and 2442 hours of data, respectively. Some key statistics of these data sets is given in Table A.1.

Title	VoxCeleb1	VoxCeleb2
# Speakers	1251	6112
# Male Speakers	690	3761
# Videos	22496	150480
# hours	352	2442
# utterances	153516	1128246
Videos per Speaker	18	25
Utterances per Speaker	116	185
Avg. Utterance Length (s)	8.2	7.8

Table A.1 The VoxCeleb Datasets

A.2 LibriSpeech

The LibriSpeech data set (Panayotov et al., 2015) is a large-scale data set with nearly 1000 hours of read English speech sampled at 16 kHz. The content is derived from audiobooks from the LibriVox project. There are two subsets in either the dev set or the test set. "clean" refers to the partition of the data with low Word Error Rate (WER) speakers, whereas "other" refers to the partition of the data with high WER speakers, based on an existing model trained on another read speech data set. Some key statistics of these data sets is given in Table A.2.

	train	dev		test	
		clean	other	clean	other
# utterance	281.2k	2.7k	2.9k	2.6k	2.9k
total duration (h)	960.9	5.4	5.1	5.4	5.3
duration per utterance (s)	12.3	7.2	6.4	7.4	6.5
total # words	9403.6k	54.4k	50.9k	52.6k	52.3k
# words per utterance	33.4	20.1	17.8	20.1	17.8
# unique words	89.1k	8.3k	7.4k	8.1k	7.6k
# speakers	2338	40	33	40	33
# speakers in training set	–	0	0	0	0

Table A.2 LibriSpeech data set. The dev and test sets follow the official split. Unique words counts "Allen" and "Allen's" as two words.

For language modelling on the LibriSpeech data set, there is a separate text corpus available, which is much larger than the amount of training text. The additional text data has around 810 million words and 900 thousand unique words.

A.3 Switchboard

The Switchboard data set (Godfrey and Holliman, 1993) (Switchboard-1 release 2) is a conversational telephony data set with around 2400 two-sided recordings of landline telephone conversations sampled at 8 kHz. Each phone call is between two English speakers, and the topic of each conversation is picked from a list of 70 topics.

The Hub5 2000 evaluation data (Hub5'00) (LDC, 2002a) and its transcripts (LDC, 2002b) are used as the test set. Hub5'00 has two parts, SwitchBoard (SWB) and CallHome (CH), where each part has 20 conversations. The SWB part has some overlapping speakers with the training set because it was first collected together with the training set but unreleased. The

CH part is from the CH English corpus (Canavan et al., 1997). Note that although the two parts are both landline telephone conversations, they are different in nature. The two callers in SWB did not know each other, and they followed the assigned topics during phone calls. However, CH participants in each call were family or friends, which resulted in less topicality and formality (Fiscus et al., 2000). In addition, CH contains more accented speech. Therefore, the CH part is expected to be more challenging.

	train	train-no-overlap	SWB	CH
# utts	249.3k	244.6k	1831	2627
total duration (hrs)	311.25	305.2	2.05	1.55
avg. duration (s)	4.5	4.03	2.13	
# words	3081.3k	3023.1k	20.5k/19.9k	21.1k/20.7k
avg. # words	12.4	12.4	11.2/10.9	8.0/7.9
# unique words	30.1k	29.9k	2.3k/2.1k	2.3k/2.1k
# speakers	520	484	40	40

Table A.3 Switchboard-1 real ease-2 dataset (LDC97S62). The SWB and CH portions are from the Hub5'00 data set (LDC2002S09). The eight unscored utterances from CH are not included in these statistics. Hesitations are excluded. For SWB and CH the number of words is given with and without partial words.

For language modelling on the SWB dataset, apart from the corresponding transcriptions of the 300-hour training data, sometimes Fisher (FSH) transcriptions (Cieri et al., 2004a, 2005) is also included as additional in-domain text data to improve the Language Model (LM). The FSH text has around 22 million words and 65 thousand unique words.

References

Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Chen, J., Chrzanowski, M., Coates, A., Diamos, G.F., Elsen, E., Engel, J., Fan, L.J., Fougner, C., Hannun, A.Y., Jun, B., Han, T.X., LeGresley, P., Li, X., Lin, L., Narang, S., Ng, A., Ozair, S., Prenger, R.J., Qian, S., Raiman, J., Satheesh, S., Seetapun, D., Sengupta, S., Sriram, A., Wang, C.J., Wang, Y., Wang, Z., Xiao, B., Xie, Y., Yogatama, D., Zhan, J., and Zhu, Z. (2016). Deep Speech 2 : End-to-end speech recognition in English and Mandarin. *Proc. ICML*, New York, USA. (Cited on page 93.)

Ardila, R., Branson, M., Davis, K., Kohler, M., Meyer, J., Henretty, M., Morais, R., Saunders, L., Tyers, F., and Weber, G. (2020). Common voice: A massively-multilingual speech corpus. *Proc. LREC*, Marseille, France. (Cited on page 76.)

Ashby, M. and Maidment, J. (2005). *Introducing phonetic science*. Cambridge University Press. (Cited on page 43.)

Audhkhasi, K., Georgiou, P.G., and Narayanan, S.S. (2012). Analyzing quality of crowd-sourced speech transcriptions of noisy audio for acoustic model adaptation. *Proc. ICASSP*, Kyoto, Japan. (Cited on page 95.)

Ba, J.L., Kiros, J.R., and Hinton, G.E. (2016). Layer normalization. *Proc. NIPS Deep Learning Symposium*, Barcelona, Spain. (Cited on page 19.)

Babu, A., Wang, C., Tjandra, A., Lakhotia, K., Xu, Q., Goyal, N., Singh, K., von Platen, P., Saraf, Y., Pino, J., et al. (2021). Xls-r: Self-supervised cross-lingual speech representation learning at scale. *arXiv:2111.09296*. (Cited on page 88.)

Baevski, A., Hsu, W.N., Xu, Q., Babu, A., Gu, J., and Auli, M. (2022). Data2vec: A general framework for self-supervised learning in speech, vision and language. *Proc. ICML*, Baltimore, USA. (Cited on pages 85, 153, and 180.)

Baevski, A., Schneider, S., and Auli, M. (2020a). vq-wav2vec: Self-supervised learning of discrete speech representations. *Proc. ICLR*, Addis Ababa, Ethiopia. (Cited on pages 87 and 153.)

Baevski, A., Zhou, H., rahman Mohamed, A., and Auli, M. (2020b). wav2vec 2.0: A framework for self-supervised learning of speech representations. *Proc. NeurIPS*, Vancouver, Canada. (Cited on pages 87 and 153.)

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *Proc. ICLR*, Banff, Canada. (Cited on page 17.)

Bahl, L., Bakis, R., Jelinek, F., and Mercer, R. (1980). Language-model/acousticchannel-model balance mechanism. *IBM Technical Disclosure Bulletin*, Vol. 23:3464-3465. (Cited on page 53.)

Bell, P., Gales, M.J.F., Hain, T., Kilgour, J., Lanchantin, P., Liu, X., McParland, A., Renals, S., Saz, O., Wester, M., and Woodland, P.C. (2015). The mgb challenge: Evaluating multi-genre broadcast media recognition. *Proc. ASRU*, Scottsdale, Arizona. (Cited on page 96.)

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. *Proc. NIPS*, Montreal, Canada. (Cited on page 71.)

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, Vol. 5, pp. 157–166. (Cited on page 16.)

Bilmes, J. (2008). Fisher kernels for DBNs. , Vol. Tech. Rep. (Cited on page 100.)

Bishop, C.M. (1994). Mixture density networks. Technical report, Computing Research Group, Department of Computer Science, Aston University. (Cited on page 11.)

Bishop, C.M. (2006). *Pattern recognition and machine learning*. Springer. (Cited on page 23.)

Blum, A. and Chawla, S. (2001). Learning from labeled and unlabeled data using graph mincuts. *Proc. ICML*, Williamstown, USA. (Cited on page 77.)

Bourlard, H.A. and Morgan, N. (1994). *Connectionist speech recognition: A hybrid approach*. Springer. (Cited on pages 154 and 166.)

Canavan, A., Graff, D., and Zipperlen, G. (1997). CALLHOME American English speech LDC97S42. *Web Download. Philadelphia: Linguistic Data Consortium*. (Cited on page 185.)

Caron, M., Bojanowski, P., Joulin, A., and Douze, M. (2018). Deep clustering for unsupervised learning of visual features. *Proc. ECCV*, Munich, Germany. (Cited on pages 84 and 88.)

Castelli, V. and Cover, T.M. (1996). The relative value of labeled and unlabeled samples in pattern recognition with an unknown mixing parameter. *IEEE Transactions on Information Theory*, Vol. 42, No. 6. (Cited on pages 74 and 117.)

Chan, H.Y. and Woodland, P. (2004). Improving broadcast news transcription by lightly supervised discriminative training. *Proc. ICASSP*, Montreal, Canada. (Cited on page 81.)

Chan, W., Jaitly, N., Le, Q.V., and Vinyals, O. (2016). Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. *Proc. ICASSP*, Shanghai, China. (Cited on pages 17, 34, and 69.)

Chan, W., Park, D., Lee, C., Zhang, Y., Le, Q., and Norouzi, M. (2021). Speechstew: Simply mix all available speech recognition data to train one large neural network. *arXiv preprint arXiv:2104.02133*. (Cited on pages 76 and 93.)

Chapelle, O., Scholkopf, B., and Zien, A. (2006). *Semi-Supervised Learning*. MIT Press. (Cited on page 73.)

Chen, N.F., Ni, C., Chen, I., Sivadas, S., Pham, V.T., Xu, H., Xiao, X., Lau, T.S., Leow, S.J., Lim, B.P., Leung, C., Wang, L., Lee, C., Goh, A., Chng, E.S., Ma, B., and Li, H. (2015). Low-resource keyword search strategies for Tamil. *Proc. ICASSP*, Brisbane, Australia. (Cited on page 102.)

Chen, S., Wang, C., Chen, Z., Wu, Y., Liu, S., Chen, Z., Li, J., Kanda, N., Yoshioka, T., Xiao, X., Wu, J., Zhou, L., Ren, S., Qian, Y., Qian, Y., Wu, J., Zeng, M., Yu, X., and Wei, F. (2022). WavLM: Large-scale self-supervised pre-training for full stack speech processing. *IEEE Journal of Selected Topics in Signal Processing*, Vol. 16, No. 6, pp. 1505–1518. (Cited on pages 88, 119, 153, 158, 164, and 180.)

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020). A simple framework for contrastive learning of visual representations. *Proc. ICML*, Virtual. (Cited on pages 84, 85, and 180.)

Chiu, C.C., Sainath, T.N., Wu, Y., Prabhavalkar, R., Nguyen, P., Chen, Z., Kannan, A., Weiss, R.J., Rao, K., Gonina, E., Jaitly, N., Li, B., Chorowski, J., and Bacchiani, M. (2018). State-of-the-art speech recognition with sequence-to-sequence models. *Proc. ICASSP*, Calgary, Canada. (Cited on pages 7 and 34.)

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *Proc. EMNLP*, Doha, Qatar. (Cited on page 15.)

Chorowski, J., Bahdanau, D., Serdyuk, D., Cho, K., and Bengio, Y. (2015). Attention-based models for speech recognition. *Proc. NIPS*, Montreal, Canada. (Cited on page 34.)

Chorowski, J. and Jaitly, N. (2017). Towards better decoding and language model integration in sequence to sequence models. *Proc. Interspeech*, Stockholm, Sweden. (Cited on page 70.)

Chung, J.S., Nagrani, A., and Zisserman, A. (2018). Voxceleb2: Deep speaker recognition. *Proc. Interspeech*, Hyderabad, India. (Cited on pages 114 and 183.)

Chung, Y., Hsu, W., Tang, H., and Glass, J.R. (2019). An unsupervised autoregressive model for speech representation learning. *Proc. Interspeech*, Graz, Austria. (Cited on page 153.)

Cieri, C., Graff, D., Kimball, O., Miller, D., and Walker, K. (2004a). Fisher English training speech part 1 transcripts LDC2004T19. *Web Download. Philadelphia: Linguistic Data Consortium*. (Cited on page 185.)

Cieri, C., Graff, D., Kimball, O., Miller, D., and Walker, K. (2005). Fisher English training speech part 2 transcripts LDC2005T19. *Web Download. Philadelphia: Linguistic Data Consortium*. (Cited on page 185.)

Cieri, C., Miller, D., and Walker, K. (2004b). The Fisher corpus: A resource for the next generations of speech-to-text. *Proc. LREC*, Lisbon, Portugal. (Cited on pages 95 and 145.)

Cieri, C. and Strassel, S. (2007). Closer still to a robust, all digital, empirical, reproducible sociolinguistic methodology. *Proc. NWAV 38*, Ottawa, Canada. (Cited on pages xix and 95.)

Cozman, F.G., Cohen, I., and Cirelo, M.C. (2003). Semi-supervised learning of mixture models. *Proc. ICML*, Washington DC., USA. (Cited on page 74.)

Cyrta, P., Trzciński, T., and Stokowiec, W. (2017). Speaker diarization using deep recurrent convolutional neural networks for speaker embeddings. *Proc. ISAT*, Szklarska Poręba, Poland. (Cited on page 110.)

Dagan, I. and Engelson, S.P. (1995). Committee-based sampling for training probabilistic classifiers. *Proc. ICML*, Tahoe City, USA. (Cited on page 98.)

Dahl, G.E., Yu, D., Deng, L., and Acero, A. (2012). Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech, & Language Processing*, Vol. 20, No. 1, pp. 30–42. (Cited on pages 7, 64, 154, and 166.)

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q.V., and Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. *Proc. ACL*, Florence, Italy. (Cited on page 51.)

Davis, S. and Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, & Signal Processing*, Vol. 28, No. 4, pp. 357–366. (Cited on page 36.)

Dehak, N., Kenny, P.J., Dehak, R., Dumouchel, P., and Ouellet, P. (2011). Front-end factor analysis for speaker verification. *IEEE Transactions on Audio, Speech, & Language Processing*, Vol. 19, No. 4, pp. 788–798. (Cited on page 110.)

Demiriz, A., Bennett, K.P., and Embrechts, M.J. (1999). Semi-supervised clustering using genetic algorithms. *Proc. of Artificial Neural Networks in Engineering*. (Cited on page 81.)

Demuynck, K., Duchateau, J., Van Compernolle, D., and Wambacq, P. (2000). An efficient search space representation for large vocabulary continuous speech recognition. *Speech Communication*, Vol. 30, pp. 37–53. (Cited on page 52.)

Deng, J., Guo, J., Xue, N., and Zafeiriou, S. (2019). Arcface: Additive angular margin loss for deep face recognition. *Proc. CVPR*, Long Beach. (Cited on page 110.)

Devlin, J., Chang, M.W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. *Proc. NAACL*, Minneapolis, USA. (Cited on pages 18, 85, and 86.)

Díez, M., Burget, L., Wang, S., Rohdin, J., and Cernocký, J.H. (2019). Bayesian HMM based X-vector clustering for speaker diarization. *Proc. Interspeech*, Graz, Austria. (Cited on page 110.)

Doersch, C., Gupta, A., and Efros, A.A. (2015). Unsupervised visual representation learning by context prediction. *Proc. ICCV*, Santiago, Chile. (Cited on page 83.)

Dong, L., Wang, F., and Xu, B. (2019). Self-attention aligner: A latency-control end-to-end model for ASR using self-attention network and chunk-hopping. *Proc. ICASSP*, Brighton, UK. (Cited on page 19.)

Dong, L., Xu, S., and Xu, B. (2018). Speech-Transformer: A no-recurrence sequence-to-sequence model for speech recognition. *Proc. ICASSP*, Calgary, Canada. (Cited on page 68.)

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, Vol. 12, pp. 2121–2159. (Cited on page 23.)

Evanini, K., Higgins, D., and Zechner, K. (2010). Using amazon mechanical turk for transcription of non-native speech. *Proc. NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, Los Angeles, USA. (Cited on page 96.)

Evermann, G. and Woodland, P.C. (2000). Posterior probability decoding, confidence estimation and system combination. *Proc. NIST Speech Transcription Workshop*, College Park, USA. (Cited on page 59.)

Fathullah, Y., Zhang, C., and Woodland, P.C. (2020). Improved large-margin softmax loss for speaker diarisation. *Proc. ICASSP*, Barcelona, Spain. (Cited on page 110.)

Ferrer-i Cancho, R. and Sole, R. (2002). Zipf's law and random texts. *Advances in Complex Systems*, Vol. 5, No. 1, pp. 1–6. (Cited on page 50.)

Fiscus, J.G., Fisher, W.M., Martin, A.F., Przybocki, M.A., and Pallett, D.S. (2000). 2000 NIST evaluation of conversational speech recognition over the telephone: English and Mandarin performance results. *Proc. NIST Speech Transcription Workshop*, College Park, USA. (Cited on page 185.)

Fousek, P., Lamel, L., and Gauvain, J.L. (2008). Transcribing broadcast data using MLP features. *Proc. Interspeech*, Brisbane, Australia. (Cited on page 47.)

Fraga-Silva, T., Gauvain, J.L., Lamel, L., Laurent, A., Le, V.B., and Messaoudi, A. (2015a). Active learning based data selection for limited resource STT and KWS. *Proc. Interspeech*, Dresden, Germany. (Cited on pages 93 and 102.)

Fraga-Silva, T., Laurent, A., Gauvain, J., Lamel, L., Le, V., and Messaoudi, A. (2015b). Improving data selection for low-resource STT and KWS. *Proc. ASRU*, Scottsdale, USA. (Cited on page 102.)

Fujii, A., Inui, K., Tokunaga, T., and Tanaka, H. (1998). Selective sampling for example-based word sense disambiguation. *Computational Linguistics*, Vol. 24, No. 4, pp. 573–597. (Cited on page 101.)

Furui, S. (1986). Speaker-independent isolated word recognition based on emphasized spectral dynamics. *Proc. ICASSP*, Tokyo, Japan. (Cited on page 36.)

Gage, P. (1994). A new algorithm for data compression. *The C Users Journal archive*, Vol. 12, No. 2, pp. 23–38. (Cited on page 38.)

Gal, Y. (2016). *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge. (Cited on pages 29 and 30.)

Gal, Y. and Ghahramani, Z. (2016). Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. *Proc. ICML*, New York City, USA. (Cited on pages 29, 31, and 124.)

Gal, Y., Hron, J., and Kendall, A. (2017a). Concrete Dropout. *Proc. NIPS*, Long Beach, USA. (Cited on pages 28 and 31.)

Gal, Y., Islam, R., and Ghahramani, Z. (2017b). Deep Bayesian active learning with image data. *Proc. ICML*, Sydney, Australia. (Cited on page 98.)

Gales, M. and Young, S. (2008). The application of hidden Markov models in speech recognition. *Foundations & Trends in Signal Processing*, Vol. 1, No. 3, pp. 195–304. (Cited on page 34.)

Gales, M.J.F. (1998). Maximum likelihood linear transformations for HMM-based speech recognition. *Computer Speech & Language*, Vol. 12, pp. 75–98. (Cited on page 77.)

Gales, M.J.F., Knill, K., and Ragni, A. (2015). Unicode-based graphemic systems for limited resource languages. *Proc. ICASSP*, Brisbane, Australia. (Cited on page 38.)

Garofolo, J.S., Lamel, L.F., Fisher, W.M., Fiscus, J.G., Pallett, D.S., Dahlgren, N.L., and Zue, V. (1993). TIMIT acoustic-phonetic continuous speech corpus LDC93S1. *Linguistic Data Consortium*. (Cited on page 107.)

Gidaris, S., Singh, P., and Komodakis, N. (2018). Unsupervised representation learning by predicting image rotations. *Proc. ICLR*, Vancouver, Canada. (Cited on page 84.)

Gillick, L. and Cox, S. (1989). Some statistical issues in the comparison of speech recognition algorithms. *Proc. ICASSP*, Glasgow, UK. (Cited on page 132.)

Glenn, M.L., Strassel, S.M., Lee, H., Maeda, K., Zakhary, R., and Li, X. (2010). Transcription methods for consistency, volume and efficiency. *Proc. LREC*, Valetta, Malta. (Cited on page 94.)

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Proc. AISTATS*, Sardinia, Italy. (Cited on pages 25 and 26.)

Godfrey, J. and Holliman, E. (1993). Switchboard-1 release 2 LDC97S62. *Linguistic Data Consortium*. (Cited on page 184.)

Goel, V. and Byrne, W.J. (2000). Minimum Bayes-risk automatic speech recognition. *Computer Speech & Language*, Vol. 14, No. 2, pp. 115–135. (Cited on page 60.)

Goodfellow, I.J., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT Press. (Cited on page 24.)

Grandvalet, Y. and Bengio, Y. (2004). Semi-supervised learning by entropy minimization. *Proc. NIPS*, Vancouver, Canada. (Cited on pages 79 and 80.)

Graves, A. (2012). Sequence transduction with recurrent neural networks. *Proc. ICML Representation Learning Workshop*, Edinburgh, UK. (Cited on pages 34 and 66.)

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv:1308.0850*. (Cited on page 17.)

Graves, A., Fernández, S., Gomez, F.J., and Schmidhuber, J. (2006). Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. *Proc. ICML*, Pittsburgh, USA. (Cited on pages 49, 66, 158, and 162.)

Graves, A., Mohamed, A., and Hinton, G.E. (2013). Speech recognition with deep recurrent neural networks. *Proc. ICASSP*, Vancouver, Canada. (Cited on page 66.)

Grezl, F., Karafiat, M., Kontar, S., and Cernocky, J. (2007). Probabilistic and bottle-neck features for LVCSR of meetings. *Proc. ICASSP*, Honolulu, USA. (Cited on page 47.)

Grill, J.B., Strub, F., Altché, F., Tallec, C., Richemond, P., Buchatskaya, E., Doersch, C., Avila Pires, B., Guo, Z., Gheshlaghi Azar, M., Piot, B., Kavukcuoglu, K., Munos, R., and Valko, M. (2020). Bootstrap your own latent - A new approach to self-supervised learning. *Proc. Neurips*, Virtual. (Cited on page 84.)

Gulati, A., Qin, J., Chiu, C.C., Parmar, N., Zhang, Y., Yu, J., Han, W., Wang, S., Zhang, Z., Wu, Y., and Pang, R. (2020). Conformer: Convolution-augmented Transformer for speech recognition. *Proc. Interspeech*, Shanghai, China. (Cited on page 68.)

Gulcehre, C., Firat, O., Xu, K., Cho, K., and Bengio, Y. (2017). On integrating a language model into neural machine translation. *Computer Speech & Language*, Vol. 45, pp. 137–148. (Cited on page 49.)

Guo, P., Boyer, F., Chang, X., Hayashi, T., Higuchi, Y., et al. (2021). Recent developments on ESPnet toolkit boosted by Conformer. *Proc. ICASSP*, Toronto, Canada. (Cited on page 68.)

Hadian, H., Sameti, H., Povey, D., and Khudanpur, S. (2018). Flat-start single-stage discriminatively trained HMM-based models for ASR. *Transactions on Audio, Speech, & Language Processing*, Vol. 26, No. 11, pp. 1949–1961. (Cited on page 49.)

Haeusser, P., Plapp, J., Golkov, V., Aljalbout, E., and Cremers, D. (2018). Associative deep clustering: Training a classification network with no labels. *Proc. German Conference on Pattern Recognition*, Stuttgart, Germany. (Cited on page 81.)

Hakkani-Tür, D., Riccardi, G., and Gorin, A. (2002). Active learning for automatic speech recognition. *Proc. ICASSP*, Orlando, USA. (Cited on pages 93, 98, and 151.)

Hamanaka, Y., Shinoda, K., Furui, S., Emori, T., and Koshinaka, T. (2010). Speech modeling based on committee-based active learning. *Proc. ICASSP*, Dallas, USA. (Cited on page 99.)

He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. (2020). Momentum contrast for unsupervised visual representation learning. *Proc. CVPR*, Virtual. (Cited on page 85.)

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Proc. ICCV*, Santiago, Chile. (Cited on page 26.)

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. *Proc. CVPR*, Las Vegas, USA. (Cited on page 19.)

Hermansky, H. (1990). Perceptual linear predictive (PLP) analysis of speech. *The Journal of the Acoustical Society of America*, Vol. 87, pp. 1738–1752. (Cited on page 36.)

Hermansky, H., Ellis, D.P.W., and Sharma, S. (2000). Tandem connectionist feature extraction for conventional HMM systems. *Proc. ICASSP*, Istanbul, Turkey. (Cited on page 47.)

Hernandez-Lobato, J.M. and Adams, R. (2015). Probabilistic backpropagation for scalable learning of Bayesian neural networks. *Proc. ICML*, Lille, France. (Cited on page 27.)

Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A.r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T., and Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, Vol. 29, No. 6, pp. 82–97. (Cited on page 7.)

Hinton, G., Osindero, S., and Teh, Y.W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, Vol. 18, No. 7, pp. 1527–54. (Cited on page 83.)

Hinton, G.E. and Shallice, T. (1991). Lesioning an attractor network: Investigations of acquired dyslexia. *Psychological review*, Vol. 98, No. 1, pp. 74. (Cited on page 7.)

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, Vol. 9, No. 8, pp. 1735–1780. (Cited on pages 15 and 16.)

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, Vol. 2, No. 5, pp. 359–366. (Cited on page 10.)

Houlsby, N., Huszár, F., Ghahramani, Z., and Lengyel, M. (2011). Bayesian active learning for classification and preference learning. *arXiv:1112.5745*. (Cited on pages 98, 121, 122, and 123.)

Howard, J. and Ruder, S. (2018). Fine-tuned language models for text classification. *Proc. ACL*, Melbourne, Australia. (Cited on page 85.)

Hsu, W.N., Bolte, B., Tsai, Y.H.H., Lakhotia, K., Salakhutdinov, R., and Mohamed, A. (2021a). Hubert: Self-supervised speech representation learning by masked prediction of hidden units. *IEEE/ACM Transactions on Audio, Speech, & Language Processing*, Vol. 29, pp. 3451–3460. (Cited on pages 88, 153, 154, 155, 158, 160, 161, 163, and 164.)

Hsu, W.N., Tsai, Y.H.H., Bolte, B., Salakhutdinov, R., and Mohamed, A. (2021b). Hubert: How much can a bad teacher benefit ASR pre-training? *Proc. ICASSP*, Toronto, Canada. (Cited on page 153.)

Huang, J., Child, R., Rao, V., Liu, H., Satheesh, S., and Coates, A. (2016). Active learning for speech recognition: the power of gradients. *Proc. Continual Learning and Deep Networks Workshop at NIPS*, Barcelona, Spain. (Cited on page 99.)

Huang, J.T. and Hasegawa-Johnson, M. (2010). Semi-supervised training of Gaussian mixture models by conditional entropy minimization. *Proc. Interspeech*, Makuhari, Japan. (Cited on page 80.)

Huang, Z., Wang, S., and Yu, K. (2018). Angular softmax for short-duration text-independent speaker verification. *Proc. Interspeech*, Hyderabad, India. (Cited on page 110.)

Inan, H., Khosravi, K., and Socher, R. (2017). Tying word vectors and word classifiers: A loss framework for language modeling. *Proc. ICLR*, Vancouver, Canada. (Cited on page 51.)

Irie, K., Zeyer, A., Schlüter, R., and Ney, H. (2019). Language modeling with deep Transformers. *Proc. Interspeech*, Graz, Austria. (Cited on page 51.)

Itoh, N., Sainath, T.N., Jiang, D.N., Zhou, J., and Ramabhadran, B. (2012). N-best entropy based data selection for acoustic modeling. *Proc. ICASSP*, Kyoto, Japan. (Cited on pages 98, 100, and 127.)

Jaakola, T. and Haussler, D. (1999). Exploiting generative models in discriminative classifiers. *Proc. NIPS*, Denver, USA. (Cited on page 100.)

Jang, E., Gu, S., and Poole, B. (2017). Categorical reparameterization with gumbel-softmax. *Proc. ICLR*, Toulon, France. (Cited on page 87.)

Jelinek, F. (1991). Up from trigrams! - the struggle for improved language models. *Proc. Eurospeech*, Genoa, Italy. (Cited on page 50.)

Ji, X., Henriques, J.F., and Vedaldi, A. (2018). Invariant information distillation for unsupervised image segmentation and clustering. *Proc. ICCV*, Instanbul, Turkey. (Cited on page 81.)

Jiang, D., Li, W., Cao, M., Zou, W., and Li, X. (2021). Speech simclr: Combining contrastive and reconstruction objective for self-supervised speech representation learning. *Proc. Interspeech*, Brno, Czech Republic. (Cited on pages 85 and 181.)

Jim, K.C., Giles, C., and Horne, B. (1996). An analysis of noise in recurrent neural networks: Convergence and generalization. *IEEE Transactions on Neural Networks*, Vol. 7, No. 6, pp. 1424–1438. (Cited on page 28.)

Jordan, M.I., Ghahramani, Z., Jaakkola, T.S., and Saul, L.K. (1999). An introduction to variational methods for graphical models. *Machine Learning*, Vol. 37, No. 2. (Cited on page 30.)

Juang, B.. (1985). Maximum-likelihood estimation for mixture multivariate stochastic observations of Markov chains. *AT&T Technical Journal*, Vol. 64, No. 6, pp. 1235–1249. (Cited on page 45.)

Kalgaonkar, K., Liu, C., Gong, Y., and Yao, K. (2015). Estimating confidence scores on ASR results using recurrent neural networks. *Proc. ICASSP*, Brisbane, Australia. (Cited on page 61.)

Kamm, T.M. and Meyer, G.G.L. (2002). Selective sampling of training data for speech recognition. *Proc. HLT*, San Diego, USA. (Cited on page 98.)

Kamm, T.M. and Meyer, G.G.L. (2003). Word-selective training for speech recognition. *Proc. ASRU*, St Thomas, USA. (Cited on page 98.)

Kastanos, A., Ragni, A., and Gales, M.J.F. (2020). Confidence estimation for black box automatic speech recognition systems using lattice recurrent neural networks. *Proc. ICASSP*, Barcelona, Spain. (Cited on page 61.)

Katz, S. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, & Signal Processing*, Vol. 35, pp. 400–401. (Cited on page 50.)

Kemp, T. and Waibel, A. (1998). Unsupervised training of a speech recognizer using tv broadcasts. *Proc. ICSLP*, Sydney, Australia. (Cited on page 78.)

Kemp, T. and Waibel, A. (1999). Unsupervised training of a speech recognizer: Recent experiments. *Proc. EuroSpeech*, Budapest, Hungary. (Cited on page 78.)

Khorram, S., Kim, J., Tripathi, A., Lu, H., Zhang, Q., and Sak, H. (2022). Contrastive siamese network for semi-supervised speech recognition. *Proc. ICASSP*, Singapore. (Cited on pages 85 and 181.)

Kimball, O., Kao, C.L., Iyer, R., Arvizo, T., and Makhoul, J. (2004). Using quick transcriptions to improve conversational speech models. *Proc. ICSLP*, Jeju Island, South Korea. (Cited on page 95.)

Kingma, D. and Welling, M. (2014). Auto-encoding variational bayes. *Proc. ICLR*, Banff, Canada. (Cited on page 83.)

Kingma, D.P. and Ba, J. (2015). Adam: A method for stochastic optimization. *Proc. ICLR*, San Diego, USA. (Cited on pages 23 and 25.)

Kingma, D.P., Mohamed, S., Jimenez Rezende, D., and Welling, M. (2014). Semi-supervised learning with deep generative models. *Proc. NIPS*, Montreal, Canada. (Cited on page 83.)

Kneser, R. and Ney, H. (1995). Improved backing-off for M-gram language modeling. *Proc. ICASSP*, Detroit, USA. (Cited on page 50.)

Koehn, P. (2009). *Statistical machine translation*. Cambridge University Press. (Cited on page 49.)

Kosiorek, A.R., Sabour, S., Teh, Y.W., and Hinton, G.E. (2019). Stacked capsule autoencoders. *Proc. NeuRIPS*, Vancouver, Canada. (Cited on page 81.)

Kreyssig, F.L., Shi, Y., Guo, J., Sari, L., Mohamed, A., and Woodland, P.C. (2023). Biased self-supervised learning for asr. *Proc. Interspeech*, Dublin, Ireland. (Cited on page 5.)

Kreyssig, F.L. and Woodland, P.C. (2020). Cosine-distance virtual adversarial training for semi-supervised speaker-discriminative acoustic embeddings. *Proc. Interspeech*, Shanghai, China. (Cited on pages 5 and 104.)

Kreyssig, F.L., Zhang, C., and Woodland, P.C. (2018). Improved TDNNs using deep kernels and frequency dependent grid-RNNs. *Proc. ICASSP*, Calgary. (Cited on page 15.)

Kreyszig, E., Kreyszig, H., and Norminton, E.J. (2011). *Advanced engineering mathematics*, chapter 20.8 Power method for eigenvalues, pages 885–887. Wiley, Hoboken, NJ, tenth edition. (Cited on page 105.)

Krizhevsky, A., Sutskever, I., and Hinton, G.E. (2012). ImageNet classification with deep convolutional neural networks. *Proc. NIPS*, Lake Tahoe, USA. (Cited on page 7.)

Kudo, T. and Richardson, J. (2018). SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *Proc. EMNLP*, Brussels, Belgium. (Cited on page 162.)

Laine, S. and Aila, T. (2017). Temporal ensembling for semi-supervised learning. *Proc. ICLR*, Toulon, France. (Cited on page 103.)

Lakhotia, K., Kharitonov, E., Hsu, W., Adi, Y., Polyak, A., Bolte, B., Nguyen, T.A., Copet, J., Baevski, A., Mohamed, A., and Dupoux, E. (2021). On generative spoken language modeling from raw audio. *Transactions of the Association for Computational Linguistics*, Vol. 9, pp. 1336–1354. (Cited on pages 168 and 181.)

Lakshminarayanan, B., Pritzel, A.P., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. *Proc. NIPS*. (Cited on page 124.)

Lamel, L., Gauvain, J., and Adda, G. (2002a). Unsupervised acoustic model training. *Proc. ICASSP*, Orlando, USA. (Cited on page 79.)

Lamel, L., Gauvain, J.l., and Adda, G. (2000). Lightly supervised acoustic model training. *Proc. ISCA ITRW ASR2000*, Paris, France. (Cited on page 81.)

Lamel, L., Gauvain, J.L., and Adda, G. (2001). Investigating lightly supervised acoustic model training. *Proc. ICASSP*, Salt Lake City, USA. (Cited on page 81.)

Lamel, L., Gauvain, J.L., and Adda, G. (2002b). Lightly supervised and unsupervised acoustic model training. *Computer Speech & Language*, Vol. 16, No. 1, pp. 115–129. (Cited on page 81.)

Lang, K.J. and Baum, E.B. (1992). Query learning can work poorly when a human oracle is used. *Proc. IEEE IJCNN*, Baltimore, USA. (Cited on page 91.)

LDC (2002a). 2000 HUB5 English evaluation speech LDC2002S09. *Web Download. Philadelphia: Linguistic Data Consortium*. (Cited on page 184.)

LDC (2002b). 2000 HUB5 English evaluation transcripts LDC2002T43. *Web Download. Philadelphia: Linguistic Data Consortium*. (Cited on page 184.)

Le, D., Zhang, X., Zheng, W., Fügen, C., Zweig, G., and Seltzer, M.L. (2019). From senones to chenones: Tied context-dependent graphemes for hybrid speech recognition. *Proc. ASRU*, Singapore. (Cited on page 38.)

LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., and Jackel, L.D. (1989). Backpropagation applied to handwritten ZIP code recognition. *Neural Computation*, Vol. 1, No. 4, pp. 541–551. (Cited on page 15.)

Lee, A., Chen, P.J., Wang, C., Gu, J., Popuri, S., Ma, X., Polyak, A., Adi, Y., He, Q., Tang, Y., Pino, J., and Hsu, W.N. (2022a). Direct speech-to-speech translation with discrete units. *Proc. ACL*, Dublin, Ireland. (Cited on page 181.)

Lee, A., Gong, H., Duquenne, P., Schwenk, H., Chen, P., Wang, C., Popuri, S., Pino, J.M., Gu, J., and Hsu, W. (2022b). Textless speech-to-speech translation on real data. *Proc. NAACL*, Seattle, USA. (Cited on pages 168 and 181.)

Lee, D. (2013). Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. *Proc. ICML Workshop on Challenges in Representation Learning*, Atlanta, USA. (Cited on page 78.)

Leggetter, C. and Woodland, P. (1995). Maximum likelihood linear regression for speaker adaptation of continuous density hidden Markov models. *Computer Speech & Language*, Vol. 9, No. 2, pp. 171–185. (Cited on page 77.)

Lewis, D.D. and Catlett, J. (1994). Heterogeneous uncertainty sampling for supervised learning. *Proc. ICML*, New Brunswick, USA. (Cited on page 97.)

Li, B., Gulati, A., Yu, J., Sainath, T.N., Chiu, C., et al. (2021a). A better and faster end-to-end model for streaming ASR. *Proc. ICASSP*, Toronto, Canada. (Cited on page 67.)

Li, B., Gulati, A., Yu, J., Sainath, T.N., Chiu, C.C., Narayanan, A., Chang, S.Y., Pang, R., He, Y., Qin, J., Han, W., Liang, Q., Zhang, Y., Strohman, T., and Wu, Y. (2021b). A better and faster end-to-end model for streaming ASR. *Proc. ICASSP*, Toronto (virtual), Canada. (Cited on pages 19 and 68.)

Li, Q., Kreyssig, F.L., Zhang, C., and Woodland, P.C. (2021c). Discriminative neural clustering for speaker diarisation. *Proc. SLT*, Shenzhen, China. (Cited on page 110.)

Li, Y., Gao, F., Ou, Z., and Sun, J. (2018). Angular softmax loss for end-to-end speaker verification. *Proc. ISCSLP*, Taipei City, Taiwan. (Cited on page 110.)

Lin, H. and Bilmes, J. (2009). How to select a good training-data subset for transcription: Submodular active selection for sequences. *Proc. Interspeech*, Brighton, UK. (Cited on pages 100 and 101.)

Lin, H. and Bilmes, J. (2011). A Class of Submodular Functions for Document Summarization. *Proc. ACL*, Portland, USA. (Cited on page 93.)

Liu, W., Wen, Y., Yu, Z., Li, M., Raj, B., and Song, L. (2017). Sphereface: Deep hypersphere embedding for face recognition. *Proc. CVPR*, Honolulu, USA. (Cited on pages 110, 112, and 115.)

Lloyd, S. (1982). Least squares quantization in pcm. *IEEE Transactions on Information Theory*, Vol. 28, No. 2, pp. 129–137. (Cited on page 154.)

Locatello, F., Bauer, S., Lucic, M., Raetsch, G., Gelly, S., Schölkopf, B., and Bachem, O. (2019). Challenging common assumptions in the unsupervised learning of disentangled representations. *Proc. ICML*, Long Beach, USA. (Cited on page 82.)

Luo, Y., Zhu, J., Li, M., Ren, Y., and Zhang, B. (2018). Smooth neighbors on teacher graphs for semi-supervised learning. *Proc. CVPR*, Salt Lake City, USA. (Cited on page 103.)

Luu, C., Bell, P., and Renals, S. (2020). Dropclass and dropadapt: Dropping classes for deep speaker representation learning. *Proc. Speaker Odyssey*, Tokyo, Japan. (Cited on page 110.)

MacKay, D. (1992). Information-based objective functions for active data selection. *Neural Computation*, Vol. 4(4):590–604. (Cited on page 92.)

Mangu, L., Brill, E., and Stolcke, A. (2000). Finding consensus in speech recognition: Word error minimization and other applications of confusion networks. *Computer Speech & Language*, Vol. 14, pp. 373–400. (Cited on page 59.)

Manning, C.D. and Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT Press. (Cited on page 50.)

Manohar, V., Hadian, H., Povey, D., and Khudanpur, S. (2018). Semi-supervised training of acoustic models using lattice-free MMI. *Proc. ICASSP*, Calgary, Canada. (Cited on page 80.)

Manohar, V., Povey, D., and Khudanpur, S. (2015). Semi-supervised maximum mutual information training of deep neural network acoustic models. *Proc. Interspeech*, Dresden, Germany. (Cited on pages 46, 80, and 116.)

McCallum, A. and Nigam, K. (1998). Employing EM and pool-based active learning for text classification. *Proc. ICML*, San Francisco, USA. (Cited on pages 98 and 100.)

Melville, P., Yang, S.M., Saar-Tsechansky, M., and Mooney, R. (2005). Active learning for probability estimation using jensen-shannon divergence. *Proc. ECML*, Porto, Portugal. (Cited on page 99.)

Miao, Y., Gowayyed, M.A., Na, X., Ko, T., Metze, F., and Waibel, A.H. (2016). An empirical exploration of CTC acoustic models. *Proc. ICASSP*, Shanghai, China. (Cited on page 93.)

Mikolov, T. (2012). *Statistical language models based on neural networks*. PhD thesis, Brno University of Technology. (Cited on page 25.)

Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. *Proc. Interspeech*, Makuhari, Japan. (Cited on page 51.)

Miyato, T., Maeda, S., Ishii, S., and Koyama, M. (2019). Virtual adversarial training: A regularization method for supervised and semi-supervised learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 41, No. 8, pp. 1979–1993. (Cited on pages 82, 103, and 107.)

Mohamed, A., Lee, H.y., Borgholt, L., Havtorn, J.D., Edin, J., Igel, C., Kirchhoff, K., Li, S.W., Livescu, K., Maaløe, L., Sainath, T.N., and Watanabe, S. (2022). Self-supervised speech representation learning: A review. *IEEE Journal of Selected Topics in Signal Processing*. (Cited on pages 73 and 153.)

Mohamed, A., Okhonko, D., and Zettlemoyer, L. (2019). Transformers with convolutional context for ASR. *arXiv:1904.11660*. (Cited on page 19.)

Mohri, M., Pereira, F., and Riley, M. (2002). Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, Vol. 16, pp. 69–88. (Cited on pages 52 and 55.)

Mohri, M., Pereira, F., and Riley, M. (2008). Speech recognition with weighted finite-state transducers. *Proc. Springer Handbook of Speech Processing*, pages 559–584. Springer. (Cited on pages 52 and 55.)

Moritz, N., Hori, T., and Le, J. (2020). Streaming automatic speech recognition with the Transformer model. *Proc. ICASSP*, Barcelona (virtual), Spain. (Cited on page 19.)

Nagrani, A., Chung, J.S., Xie, W., and Zisserman, A. (2019). Voxceleb: Large-scale speaker verification in the wild. *Computer Science and Language*. (Cited on pages 114 and 183.)

Nagrani, A., Chung, J.S., and Zisserman, A. (2017). Voxceleb: A large-scale speaker identification dataset. *Proc. Interspeech*, Stockholm, Sweden. (Cited on pages 114 and 183.)

Noroozi, M. and Favaro, P. (2016). Unsupervised learning of visual representations by solving jigsaw puzzles. *Proc. ECCV*, Amsterdam, The Netherlands. (Cited on page 84.)

Novotney, S. and Callison-Burch, C. (2010). Cheap, fast and good enough: Automatic speech recognition with non-expert transcription. *Proc. HLT*, Riga, Latvia. (Cited on page 95.)

Odell, J.J. (1999). Network and language models for use in a speech recognition system. US Patent 6,668,243. (Cited on page 52.)

Odell, J.J., Valtchev, V., Woodland, P.C., and Young, S.J. (1994). A one pass decoder design for large vocabulary recognition. *Proc. HLT*, Plainsboro, USA. (Cited on page 52.)

Oliver, A., Odena, A., Raffel, C., Cubuk, E.D., and Goodfellow, I.J. (2018). Realistic evaluation of deep semi-supervised learning algorithms. *Proc. NeurIPS*, Montreal, Canada. (Cited on pages 73, 74, and 103.)

O'Neill, T.J. (1978). Normal discrimination with unclassified observations. *Journal of the American Statistical Association*, Vol. 73, No. 364, pp. 821–826. (Cited on pages 74 and 117.)

Oord, A.v.d., Li, Y., and Vinyals, O. (2018). Representation learning with contrastive predictive coding. *arXiv:1807.03748*. (Cited on page 153.)

Pallet, D., Fisher, W., and Fiscus, J. (1990). Tools for the analysis of benchmark speech recognition tests. *Proc. ICASSP*, Albuquerque, USA. (Cited on page 132.)

Panayotov, V., Chen, G., Povey, D., and Khudanpur, S. (2015). Librispeech: An ASR corpus based on public domain audio books. *Proc. ICASSP*, Brisbane, Australia. (Cited on pages 76, 159, and 184.)

Parent, G. and Eskenazi, M. (2011). Speaking to the crowd: looking at past achievements in using crowdsourcing for speech and predicting future challenges. *Proc. Interspeech*, Florence, Italy. (Cited on page 96.)

Park, D.S., Chan, W., Zhang, Y., Chiu, C.C., Zoph, B., Cubuk, E.D., and Le, Q. (2019). Specaugment: A simple data augmentation method for automatic speech recognition. *Proc. Interspeech*, Graz, Austria. (Cited on page 162.)

Park, D.S., Zhang, Y., Jia, Y., Han, W., Chiu, C.C., Li, B., Wu, Y., and Le, Q.V. (2020). Improved noisy student training for automatic speech recognition. *Proc. Interspeech*, Shanghai, China. (Cited on pages 79 and 165.)

Park, J., Diehl, F., Gales, M., Tomalin, M., and Woodland, P. (2011). The efficient incorporation of MLP features into automatic speech recognition systems. *Computer Speech & Language*, Vol. 25, No. 3, pp. 519–534. (Cited on page 47.)

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. *Proc. ICML*, Atlanta, USA. (Cited on page 25.)

Pascual, S., Ravanelli, M., Serrà, J., Bonafonte, A., and Bengio, Y. (2019). Learning problem-agnostic speech representations from multiple self-supervised tasks. *Proc. Interspeech*, Graz, Austria. (Cited on pages 83 and 153.)

Pearlmutter, B.A. (1994). Fast exact multiplication by the hessian. *Neural Computation*, Vol. 6, No. 1, pp. 147–160. (Cited on page 106.)

Peddinti, V., Povey, D., and Khudanpur, S. (2015). A time delay neural network architecture for efficient modeling of long temporal contexts. *Proc. Interspeech*, Dresden, Germany. (Cited on pages 14 and 115.)

Peddinti, V., Wang, Y., Povey, D., and Khudanpur, S. (2017). Low latency acoustic modeling using temporal convolution and LSTMs. *IEEE Signal Processing Letters*, Vol. 25, No. 3, pp. 373–377. (Cited on page 48.)

Peters, M.E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *Proc. NAACL*, New Orleans, USA. (Cited on page 85.)

Pham, N.Q., Nguyen, T.S., Niehues, J., Müller, M., Stüker, S., et al. (2019). Very deep self-attention networks for end-to-end speech recognition. *Proc. Interspeech*, Graz, Austria. (Cited on page 68.)

Pineda, F.J. (1987). Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, Vol. 59, pp. 2229. (Cited on page 15.)

Polyak, A., Adi, Y., Copet, J., Kharitonov, E., Lakhotia, K., Hsu, W.N., Mohamed, A., and Dupoux, E. (2021). Speech resynthesis from discrete disentangled self-supervised representations. *Proc. Interspeech*, Brno, Czech Republic. (Cited on page 181.)

Polyak, B. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, Vol. 4, No. 5, pp. 1–17. (Cited on page 25.)

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., et al. (2011). The Kaldi speech recognition toolkit. *Proc. ASRU*, Waikoloa, USA. (Cited on page 134.)

Povey, D., Hadian, H., Ghahremani, P., Li, K., and Khudanpur, S. (2018). A time-restricted self-attention layer for ASR. *Proc. ICASSP*, Calgary, Canada. (Cited on page 19.)

Povey, D., Peddinti, V., Galvez, D., Ghahremani, P., Manohar, V., et al. (2016). Purely sequence-trained neural networks for ASR based on lattice-free MMI. *Proc. Interspeech*, San Francisco, USA. (Cited on pages 48, 65, and 134.)

Press, O. and Wolf, L. (2017). Using the output embedding to improve language models. *Proc. EMNLP*, Valencia, Spain. (Cited on page 51.)

Pundak, G. and Sainath, T.N. (2016). Lower frame rate neural network acoustic models. *Proc. Interspeech*, San Francisco, USA. (Cited on page 47.)

Rabiner, L. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, Vol. 77, No. 2, pp. 257–286. (Cited on page 34.)

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training. Technical report, OpenAI. (Cited on page 85.)

Radmard, P. (2022). Deep acquisition functions for active learning. Master's thesis, University of Cambridge. (Cited on page 151.)

Ragni, A., Li, Q., Gales, M.J., and Wang, Y. (2018). Confidence estimation and deletion prediction using bidirectional recurrent neural networks. *Proc. SLT*, Athens, Greece. (Cited on pages 61 and 79.)

Rao, K., Sak, H., and Prabhavalkar, R. (2017). Exploring architectures, data and units for streaming end-to-end speech recognition with RNN-transducer. *Proc. ASRU*, Okinawa, Japan. (Cited on page 67.)

Rasmus, A., Berglund, M., Honkala, M., Valpola, H., and Raiko, T. (2015). Semi-supervised learning with ladder networks. *Proc. NIPS*, Montreal, Canada. (Cited on page 82.)

Rasmussen, C.E. and Williams, C.K.I. (2006). *Gaussian processes for machine learning*. MIT Press. (Cited on page 124.)

Ravanelli, M., Zhong, J., Pascual, S., Swietojanski, P., Monteiro, J., Trmal, J., and Bengio, Y. (2020). Multi-task self-supervised learning for robust speech recognition. *Proc. ICASSP*, Barcelona, Spain. (Cited on pages 83 and 153.)

Riccardi, G. and Hakkani-Tur, D. (2005). Active learning: Theory and applications to automatic speech recognition. *IEEE Transactions on Speech and Audio Processing*, Vol. 13, No. 4. (Cited on page 98.)

Robinson, D.W. and Dadson, R.S. (1956). A re-determination of the equal-loudness relations for pure tones. *British Journal of Applied Physics*, Vol. 7, pp. 166–181. (Cited on page 36.)

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, Vol. 65, No. 6, pp. 386. (Cited on page 9.)

Rousu, J. and Shawe-Taylor, J. (2007). Efficient computation of gapped substring kernels for large alphabets. *Journal of Machine Learning Research*, Vol. 6, pp. 1323–1344. (Cited on page 100.)

Rumelhart, D.E., Hinton, G.E., Williams, R.J., et al. (1988). Learning representations by back-propagating errors. *Cognitive Modeling*, Vol. 5, pp. 1. (Cited on page 15.)

Sainath, T., Weiss, R.J., Wilson, K., Senior, A.W., and Vinyals, O. (2015). Learning the speech front-end with raw waveform cldnns. *Proc. Interspeech*, Dresden, Germany. (Cited on page 35.)

Sajjadi, M., Javanmardi, M., and Tasdizen, T. (2016). Regularization with stochastic transformations and perturbations for deep semi-supervised learning. *Proc. NIPS*, Barcelona, Spain. (Cited on page 103.)

Sak, H., Senior, A., Rao, K., and Beaufays, F. (2015a). Fast and accurate recurrent neural network acoustic models for speech recognition. *Proc. Interspeech*, Dresden, Germany. (Cited on page 49.)

Sak, H., Senior, A., Rao, K., Irsoy, O., Graves, A., et al. (2015b). Learning acoustic frame labeling for speech recognition with recurrent neural networks. *Proc. ICASSP*, Brisbane, Australia. (Cited on page 49.)

Saon, G., Kurata, G., Sercu, T., Audhkhasi, K., Thomas, S., Dimitriadis, D., Cui, X., Ramabhadran, B., Picheny, M., Lim, L.L., Roomi, B., and Hall, P. (2017). English conversational telephone speech recognition by humans and machines. *Proc. Interspeech*, Stockholm, Sweden. (Cited on page 93.)

Saon, G., Soltau, H., Emami, A., and Picheny, M. (2014). Unfolded recurrent neural networks for speech recognition. *Proc. Interspeech*, Singapore. (Cited on page 47.)

Saon, G., Tüske, Z., Bolaños, D., and Kingsbury, B. (2021). Advancing RNN transducer technology for speech recognition. *Proc. ICASSP*, Toronto, Canada. (Cited on page 67.)

Schneider, S., Baevski, A., Collobert, R., and Auli, M. (2019). wav2vec: Unsupervised pre-training for speech recognition. *Proc. Interspeech*, Graz, Austria. (Cited on pages 86, 87, and 153.)

Schuster, M. and Paliwal, K.K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, Vol. 45, pp. 2673–2681. (Cited on page 15.)

Schwartz, R., Chow, Y., Kimball, O., Roucos, S., Krasner, M., and Makhoul, J. (1985). Context-dependent modeling for acoustic-phonetic recognition of continuous speech. *Proc. ICASSP*, Tampa, USA. (Cited on page 44.)

Seigel, M.S. and Woodland, P.C. (2014). Detecting deletions in ASR output. *Proc. ICASSP*, Florence, Italy. (Cited on page 79.)

Sell, G. and Garcia-Romero, D. (2014). Speaker diarization with PLDA i-vector scoring and unsupervised calibration. *Proc. SLT*, Lake Tahoe, USA. (Cited on page 110.)

Senior, A. (1994). *Off-line cursive handwriting recognition using recurrent neural networks*. PhD thesis, University of Cambridge. (Cited on page 64.)

Sennrich, R., Haddow, B., and Birch, A. (2016). Neural machine translation of rare words with subword units. *Proc. ACL*, Berlin. (Cited on page 162.)

Settles, B. and Craven, M. (2008). An analysis of active learning strategies for sequence labeling tasks. *Proc. EMNLP*. (Cited on pages 100, 126, and 144.)

Settles, B., Craven, M., and Ray, S. (2008). Multiple-instance active learning. *Proc. NIPS*, Vancouver, Canada. (Cited on page 99.)

Seung, H.S., Opper, M., and Sompolinsky, H. (1992). Query by committee. *Proc. COLT*, New York, USA. (Cited on page 98.)

Shannon, C.E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, Vol. 27, No. 3, pp. 379–423. (Cited on page 30.)

Shi, Y., Wang, Y., Wu, C., Yeh, C.F., Chan, J., Zhang, F., Le, D., and Seltzer, M. (2021). Emformer: Efficient memory Transformer based acoustic model for low latency streaming speech recognition. *Proc. ICASSP*, Toronto, Canada. (Cited on pages 19, 20, and 160.)

Shum, S., Dehak, N., Chuangsuwanich, E., Reynolds, D.A., and Glass, J.R. (2011). Exploiting intra-conversation variability for speaker diarization. *Proc. Interspeech*, Florence, Italy. (Cited on page 110.)

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *Proc. ICLR*, San Diego, USA. (Cited on page 7.)

Sindhwani, V., Niyogi, P., and Belkin, M. (2005). A co-regularization approach to semi-supervised learning with multiple views. *Proc. ICML*, Bonn, Germany. (Cited on page 82.)

Snyder, D., Garcia-Romero, D., Povey, D., and Khudanpur, S. (2017). Deep neural network embeddings for text-independent speaker verification. *Proc. Interspeech*, Stockholm. (Cited on page 110.)

Snyder, D., Garcia-Romero, D., Sell, G., Povey, D., and Khudanpur, S. (2018). X-vectors: Robust DNN embeddings for speaker recognition. *Proc. ICASSP*, Calgary, Canada. (Cited on pages 110 and 115.)

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, Vol. 15, pp. 1929–1958. (Cited on pages 28 and 128.)

Stafylakis, T., Rohdin, J., Plchot, O., Mizera, P., and Burget, L. (2019). Self-supervised speaker embeddings. *Proc. Interspeech*, Graz, Austria. (Cited on page 104.)

Stevens, S. (1957). On the psychophysical law. *Psychol. Rev.*, Vol. 64, pp. 153–181. (Cited on page 36.)

Sun, G., Zhang, C., and Woodland, P.C. (2019). Speaker diarisation using 2d self-attentive combination of embeddings. *Proc. ICASSP*, Brighton, UK. (Cited on page 115.)

Sun, G., Zhang, C., and Woodland, P.C. (2021). Transformer language models with LSTM-based cross-utterance information. *Proc. ICASSP*, Toronto, Canada. (Cited on page 51.)

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. *Proc. ICML*, Atlanta, USA. (Cited on page 25.)

Syed, A.R., Rosenberg, A., and Kislal, E. (2016). Supervised and unsupervised active learning for automatic speech recognition of low-resource languages. *Proc. ICASSP*, Shanghai, China. (Cited on page 101.)

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. *Proc. CVPR*, Las Vegas, USA. (Cited on page 109.)

Tieleman, T. and Hinton, G. (2012). Lecture 6.5—rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*. (Cited on page 23.)

Tüske, Z., Golik, P., Schlüter, R., and Ney, H. (2014). Acoustic modeling with deep neural networks using raw time signal for LVCSR. *Proc. Interspeech*, Singapore. (Cited on page 35.)

Valpola, H. (2015). From neural PCA to deep unsupervised learning. *Proc. Advances in Independent Component Analysis and Learning Machines*, pages 143–171. Academic Press. (Cited on page 82.)

van Dalen, R.C., Knill, K.M., Tsiakoulis, P., and Gales, M.J.F. (2015). Improving multiple-crowd-sourced transcriptions using a speech recogniser. *Proc. ICASSP*, Brisbane, Australia. (Cited on page 96.)

Variani, E., Lei, X., McDermott, E., Moreno, I.L., and Gonzalez-Dominguez, J. (2014). Deep neural networks for small footprint text-dependent speaker verification. *Proc. ICASSP*, Florence, Italy. (Cited on page 110.)

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *Proc. NIPS*, Long Beach, USA. (Cited on pages 18, 19, 68, 87, 154, and 160.)

Villalba, J., Chen, N., Snyder, D., Garcia-Romero, D., McCree, A., Sell, G., Borgstrom, J., García-Perera, L.P., Richardson, F., Dehak, R., Torres-Carrasquillo, P.A., and Dehak, N. (2020). State-of-the-art speaker recognition with neural network embeddings in NIST SRE18 and speakers in the wild evaluations. *Computer Speech & Language*, Vol. 60, pp. 101026. (Cited on page 110.)

Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, Vol. 13, pp. 260–269. (Cited on page 46.)

von Mises, R. and Pollaczek-Geiringer, H. (1929). Praktische verfahren der gleichungsauflösung. *Zeitschrift für Angewandte Mathematik und Mechanik*, Vol. 9, pp. 152–164. (Cited on page 105.)

von Platen, P., Zhang, C., and Woodland, P.C. (2019). Multi-span acoustic modelling using raw waveform signals. *Proc. Interspeech*, Graz, Austria. (Cited on page 35.)

Wahba, G. (1990). *Spline models for observational data*. Siam. (Cited on page 103.)

Waibel, A. (1989). Modular construction of time-delay neural networks for speech recognition. *Neural Computation*, Vol. 1, No. 1, pp. 39–46. (Cited on page 14.)

Waibel, A.H., Hanazawa, T., Hinton, G.E., Shikano, K., and Lang, K.J. (1989). Phoneme recognition using time-delay neural networks. *IEEE Transactions Acoustics, Speech, & Signal Processing*, Vol. 37, pp. 328–339. (Cited on page 14.)

Wallington, E., Kershenbaum, B., Klejch, O., and Bell, P. (2021). On the learning dynamics of semi-supervised training for ASR. *Proc. Interspeech*, Brno, Czech Republic. (Cited on page 78.)

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S.R. (2019). GLUE: A multi-task benchmark and analysis platform for natural language understanding. *Proc. ICLR*, New Orleans, USA. (Cited on page 7.)

Wang, C., Wang, Y., Wu, Y., Chen, S., Li, J., Liu, S., and Wei, F. (2022). Supervision-guided codebooks for masked prediction in speech pre-training. *Proc. Interspeech*, Incheon, Korea. (Cited on pages 158 and 164.)

Wang, G., Rosenberg, A., Chen, Z., Zhang, Y., and Ramabhadran, B. (2020a). Scada: Stochastic, consistent and adversarial data augmentation to improve ASR. *Proc. Interspeech*, Shanghai, China. (Cited on page 109.)

Wang, G., Rosenberg, A., Chen, Z., Zhang, Y., Ramabhadran, B., Wu, Y., and Moreno, P. (2020b). Improving speech recognition using consistent predictions on synthesized speech. *Proc. ICASSP*, Barcelona (virtual). (Cited on page 76.)

Wang, H., Ragni, A., Gales, M.J., Knill, K.M., Woodland, P.C., and Zhang, C. (2015). Joint decoding of tandem and hybrid systems for improved keyword spotting on low resource languages. *Proc. Interspeech*, Dresden, Germany. (Cited on page 47.)

Wang, L., Gales, M.J.F., and Woodland, P.C. (2007). Unsupervised Training for Mandarin Broadcast News and Conversation Transcription. *Proc. ICASSP*, Honolulu, USA. (Cited on page 76.)

Watanabe, S., Hori, T., Kim, S., Hershey, J.R., and Hayashi, T. (2017). Hybrid CTC/attention architecture for end-to-end speech recognition. *IEEE Journal of Selected Topics in Signal Processing*, Vol. 11, pp. 1240–1253. (Cited on pages 69 and 70.)

Wei, K., Liu, Y., Kirchhoff, K., and Bilmes, J. (2013). Using Document Summarization Techniques for Speech Data Subset Selection. *Proc. NAACL*, Atlanta, USA. (Cited on pages 100 and 101.)

Wei, K., Liu, Y., Kirchhoff, K., and Bilmes, J. (2014a). Unsupervised submodular subset selection for speech data. *Proc. ICASSP*. (Cited on pages 93, 101, and 102.)

Wei, K., Liu, Y., Kirchhoff, K., and Bilmes, J. (2014b). Unsupervised submodular subset selection for speech data. *Proc. ICASSP*, Florence, Italy. (Cited on page 100.)

Wen, T.H., Gasic, M., Mrksic, N., Su, P.h., Vandyke, D., and Young, S.J. (2015). Semantically conditioned LSTM-based natural language generation for spoken dialogue systems. *Proc. EMNLP*, Lisbon, Portugal. (Cited on page 7.)

Wessel, F. and Ney, H. (2001). Unsupervised training of acoustic models for large vocabulary continuous speech recognition. *Proc. ASRU*, Madonna di Campiglio, Italy. (Cited on pages 78 and 79.)

Wessel, F. and Ney, H. (2005). Unsupervised training of acoustic models for large vocabulary continuous speech recognition. *IEEE Transactions on Speech and Audio Processing*, Vol. 13, No. 1, pp. 23–31. (Cited on page 78.)

Wilpon, J.G., Rabiner, L.R., Lee, C., and Goldman, E.R. (1990). Automatic recognition of keywords in unconstrained speech using hidden Markov models. *IEEE Transactions on Acoustics, Speech, & Signal Processing*, Vol. 38, No. 11, pp. 1870–1878. (Cited on page 61.)

Woodland, P.C. and Povey, D. (2002). Large scale discriminative training of hidden Markov models for speech recognition. *Computer Speech & Language*, Vol. 16, pp. 25–47. (Cited on pages 53, 65, and 125.)

Wu, C., Wang, Y., Shi, Y., Yeh, C.F., and Zhang, F. (2020). Streaming Transformer-based acoustic models using self-attention with augmented memory. *Proc. InterSpeech*, Shanghai (virtual), China. (Cited on page 19.)

Wu, F., Kim, K., Watanabe, S., Han, K., McDonald, R., Weinberger, K.Q., and Artzi, Y. (2022). Wav2seq: Pre-training speech-to-text encoder-decoder models using pseudo languages. (Cited on page 168.)

Wu, Z., Xiong, Y., Yu, S.X., and Lin, D. (2018). Unsupervised feature learning via non-parametric instance discrimination. *Proc. CVPR*, Salt Lake City, USA. (Cited on page 85.)

Xiao, A., Zheng, W., Keren, G., Le, D., Zhang, F., Fuegen, C., Kalinli, O., Saraf, Y., and Mohamed, A. (2022). Scaling asr improves zero and few shot learning. *Proc. Interspeech*, Incheon, Korea. (Cited on page 78.)

Xu, H., Povey, D., Mangu, L., and Zhu, J. (2011). Minimum Bayes risk decoding and system combination based on a recursion for edit distance. *Computer Speech & Language*, Vol. 25, pp. 802–828. (Cited on page 60.)

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Zemel, R., and Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. *Proc. ICML*, Lille, France. (Cited on page 17.)

Yang, S.w., Chi, P.H., Chuang, Y.S., Lai, C.I.J., Lakhotia, K., Lin, Y.Y., Liu, A.T., Shi, J., Chang, X., Lin, G.T., Huang, T.H., Tseng, W.C., Lee, K.t., Liu, D.R., Huang, Z., Dong, S., Li, S.W., Watanabe, S., Mohamed, A., and Lee, H.y. (2021). Superb: Speech processing universal performance benchmark. (Cited on pages 119 and 180.)

Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised methods. *Proc. ACL*, Stroudsburg, USA. (Cited on page 78.)

Young, S., Gašić, M., Thomson, B., and Williams, J.D. (2013). Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, Vol. 101, No. 5, pp. 1160–1179. (Cited on pages 29 and 61.)

Young, S.J. (1996). Large vocabulary continuous speech recognition: A review. *IEEE Signal Processing Magazine*, Vol. 13, pp. 45–57. (Cited on page 57.)

Young, S.J., Evermann, G., Gales, M.J.F., Hain, T., Kershaw, D.J., Liu, X., Moore, G.L., Odell, J.J., Ollason, D., Povey, D., Ragni, A., Valtchev, V., Woodland, P.C., and Zhang, C. (2015). *The HTK Book (for HTK version 3.5)*. Cambridge University Engineering Department. (Cited on pages 54, 114, 115, and 128.)

Young, S.J., Odell, J.J., and Woodland, P.C. (1994). Tree-based state tying for high accuracy acoustic modelling. *Proc. HLT*, Plainsboro, USA. (Cited on page 44.)

Young, S.J., Russell, N.H., and Thornton, J.H.S. (1989). Token passing: A simple conceptual model for connected speech recognition systems. Technical report, Cambridge University Engineering Department. (Cited on page 52.)

Young, S.R. (1994). Detecting misrecognitions and out-of-vocabulary words. *Proc. ICASSP*, Adelaide, Australia. (Cited on page 61.)

Yu, D., Varadarajan, B., Deng, L., and Acero, A. (2010a). Active learning and semi-supervised learning for speech recognition: A unified framework using the global entropy reduction maximization criterion. *Computer Speech & Language*, Vol. 24, No. 3. (Cited on pages 98 and 100.)

Yu, K., Gales, M., Wang, L., and Woodland, P.C. (2010b). Unsupervised training and directed manual transcription for LVCSR. *Speech Communication*, Vol. 52, No. 7, pp. 652–663. (Cited on page 98.)

Yu, K., Gales, M.J.F., and Woodland, P.C. (2007). Unsupervised training with directed manual transcription for recognising Mandarin broadcast audio. *Proc. Interspeech*, Antwerp, Belgium. (Cited on page 98.)

Zavaliagkos, G. and Colthurst, T. (1998). Utilizing untranscribed training data to improve performance. *Proc. DARPA Broadcast News Transcription and Understanding Workshop*, Lansdowne, USA. (Cited on page 78.)

Zhai, X., Oliver, A., Kolesnikov, A., and Beyer, L. (2019). S4l: Self-supervised semi-supervised learning. *Proc. ICCV*, Seoul, South Korea. (Cited on page 103.)

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2017). Understanding deep learning requires rethinking generalization. *Proc. ICLR*, Toulon, France. (Cited on page 165.)

Zhang, C., Kreyssig, F.L., Li, Q., and Woodland, P.C. (2019). Pyhtk: Python library and ASR pipelines for HTK. *Proc. ICASSP*, Brighton, UK. (Cited on pages 115 and 128.)

Zhang, C. and Woodland, P.C. (2017). Joint optimisation of tandem systems using Gaussian mixture density neural network discriminative sequence training. *Proc. ICASSP*, New Orleans, USA. (Cited on page 47.)

Zhang, Q., Lu, H., Sak, H., Tripathi, A., McDermott, E., Koo, S., and Kumar, S. (2020). Transformer Transducer: A streamable speech recognition model with Transformer encoders and RNN-T loss. *Proc. ICASSP*, Barcelona (virtual), Spain. (Cited on page 19.)

Zhu, X., Ghahramani, Z., and Lafferty, J.D. (2003). Semi-supervised learning using Gaussian fields and harmonic functions. *Proc. ICML*, Washington DC., USA. (Cited on page 77.)

Zwicker, E. (1961). Subdivision of the audible frequency range into critical bands. *The Journal of the Acoustical Society of America*, Vol. 33, No. 2, pp. 248–248. (Cited on page 36.)