Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Simen Burud

Conversational Language Models for Low-Resource Speech Recognition

Master's thesis in Computer Science
Supervisor: Massimiliano Ruocco
Co-supervisor: Pablo Ortiz
June 2021

Simen Burud

Conversational Language Models for Low-Resource Speech Recognition

Master's thesis in Computer Science
Supervisor: Massimiliano Ruocco
Co-supervisor: Pablo Ortiz
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

Automatic Speech Recognition (ASR) systems transcribe speech to text. They have a wide range of practical applications, from dictation tools making communication much easier for people with hearing and motor impairments to low-cost indexing and search in audiovisual content. As a building block in larger machine learning systems, ASR plays a crucial role in many commercial products, such as digital voice assistants.

Many modern ASR systems are implemented as (almost) purely data-driven, end-to-end Deep Learning models. These systems show impressive results in many domains, comparable to or even surpassing human performance. Unfortunately, these techniques often struggle when tasked with transcribing low-resource languages, especially in real-life situations. Despite the term “end-to-end”, they end up relying heavily on both an external language model and a large beam search to achieve decent results.

Pre-trained attention models such as BERT (Bidirectional Encoder Representations from Transformers) have advanced state-of-the-art across many natural language processing tasks in the past few years. Several ways of integrating BERT-like models in speech recognition systems have been proposed. However, research so far have been limited to high-resource domains.

Turning our attention to low-resource domains, we introduce a data-efficient fine-tuning strategy for BERT. BERT learns to effectively use conversational context to rescore beam search results by teaching it to disambiguate good and bad transcripts. We show how this improves performance over a robust baseline system in two distinct, specialized domains: formal parliamentary debates and customer service calls. These domains are low-resource both in terms of language (Norwegian) and speech/linguistic characteristics. We also test how to produce a richer variety of candidate transcripts to cover more possibilities using a diversity bonus.

Sammendrag

Automatiske talegjenkjenningsystemer transkiberer tale til tekst. Slike systemer har et bredt spekter av praktiske bruksområder, fra dikteringsverktøy som forenkler kommunikasjon for personer med hørsels - eller motoriske funksjonsnedsettelser, til å muliggjøre søk i audiovisuelt innhold. Talegjenkjenning spiller også en viktig rolle som del av større maskinlæringsystemer i kommersielle produkter som digitale personlige assistenter.

Mange moderne talegjenkjenningsystemer bygges som (tilnærmet) rent datadrevne ende-til-ende-modeller basert på dyp læring. Disse gir imponerende resultater på mange områder. Resultatene kan ofte sammenlignes med, og er i noen tilfeller enda mer nøyaktige enn, manuelle transkripsjoner gjort av mennesker. Dessverre kommer disse teknikkene ofte til kort i møte med språk og domener der det er lite data å trene på. Til tross for tilnavnet "ende-til-ende," blir de avhengig av både en ekstern språkmodell og et omfattende heuristisk søk (vanligvis beam search) for å oppnå brukbare resultater.

I senere tid har forhåndstrente språkmodeller basert på oppmerksomhet, f.eks. BERT (toveis omformerbaserte enkoder-representasjoner), gitt store fremskritt på mange oppgaver innen språkprosessering. Også for norsk språk har Nasjonalbiblioteket bygget en BERT-modell som gir svært lovende resultater. Det har blitt foreslått en rekke teknikker for å kombinere BERT-lignende språkmodeller med talegjenkjenning, men forskningen så langt har fokusert på språk og domener der store mengder treningseksempler er tilgjengelig.

Vi retter nå fokus mot "datafattige" domener, og introduserer en treningsstrategi for BERT der vi finjusterer modellen på en svært dataeffektiv måte. Dette skjer ved å trene BERT til å skille mellom gode og dårlige transkripsjoner fra den eksisterende talemodellen. På den måten lærer BERT å skåre forslagslisten fra talemodellen for å identifisere den beste transkripsjonen. I tillegg tester vi teknikker for å tvinge talemodellen til å generere en mer mangfoldig forslagsliste.

Bruken av BERT gir betydelig bedre resultater sammenlignet med et allerede robust talegjenkjenningsystem i to spesialiserte og svært forskjellige domener: plenumsmøter i Stortinget og kundeservicesamtaler hos Telenor. Utover at det i utgangspunktet er nokså magert med norske treningsdata for talegjenkjenning, er begge disse domenene datafattige i form av lite treningsdata, distinkt språkbruk og utstrakt bruk av dialekt.

Preface

This master thesis is the final delivery of my Master of Science (MSc) degree at the Department of Computer Science at the Norwegian University of Science and Technology (NTNU). It was written as part of the long-running collaboration between NTNU and Telenor Research through the Norwegian Open AI Lab (formerly Telenor-NTNU AI Lab). I would like to thank my supervisors, Pablo Ortiz at Telenor Research and Massimiliano Ruocco at NTNU, for excellent guidance throughout the project. In addition, I would like to thank Telenor Research for the opportunity to use their compute infrastructure and datasets in my experiments.

Simen Burud
Trondheim, June 18, 2021

Contents

List of Figures

List of Tables

Acronyms

AM Acoustic Model.

ASR Automatic Speech Recognition.

BERT Bidirectional Encoder Representations from Transformers.

BS Beam Search.

CNSP Conversational Next Sentence Prediction.

HMM Hidden Markov Model.

LM Language Model.

MLM Masked Language Modeling.

NLM Neural Language Model.

NLP Natural Language Processing.

NN Artificial Neural Network.

NSP Next Sentence Prediction.

OOV out-of-vocabulary.

RNN Recurrent Neural Network.

WER Word Error Rate.

Chapter 1

Introduction

1.1 Background and motivation

The task of Automatic Speech Recognition (ASR) is to make a computer transcribe speech to text: given a segment of audio, output the text being spoken. ASR has numerous practical applications; examples include virtual voice assistants, dictation, and search in audiovisual content, among others. As an accessibility technology, ASR enables people with hearing impairments to perceive spoken announcements in public spaces and better participate in conversations with people who do not speak sign language. Combined with machine translation, it can help facilitate conversations between people who do not speak a common language.

Due to the number of practical, real-life applications, ASR has received significant research interest for many years. Since the early attempts by Davis, Biddulph, and Balashek 1952, many approaches have been proposed. For many years, research was primarily focused on Hidden Markov Model (HMM)-based pipelines. While these show impressive results and are the basis of many commercially successful ASR systems, they typically require substantial feature engineering and tuning by domain experts. In addition, the lack of capacity to propagate the error during optimization across modules leads to suboptimal results, typically in the form of sensitivity to noise and speaker variations (Kamath, J. Liu, and Whitaker 2019).

In recent years, end-to-end Deep Learning-based systems have proved to be very successful. In contrast to the HMM-based pipelines, this approach requires very little domain expertise to train: they are trained like any other neural network, taking the raw audio as input and the target transcript as output. All feature extraction and -engineering needed is learned implicitly during training.

This makes it very easy to adapt an ASR system to a new domain: “just” produce a sufficiently large training set and feed it to the model.

Even though these systems are termed end-to-end, most can be decomposed into three distinct components: the acoustic model, the decoder algorithm, and the language model. The acoustic model is primarily concerned with encoding the audio to a sequence of token probability vectors. This sequence is typically decoded using a wide, compute-intensive beam search, often guided by a language model. The top-scoring result from the beam search is often not the optimal one, indicating there is room for improvements by making better use of the decoder outputs.

ASR systems usually do not make use of context beyond the current utterance. This lack of information makes it harder to disambiguate phonetically similar transcripts, and the model often ends up outputting the statistically most common interpretation. Xiong et al. 2017 have done some work in this regard and report improvements from local conversational data.

1.1.1 Low-resource domains

As is common in Deep Learning, “sufficiently large” means thousands of hours. Synnaeve et al. 2020, for example, report continued performance improvements just by obtaining more training data, even beyond 10,000 hours of audio. For many low-resource languages and domains, obtaining such amounts of training data is prohibitively expensive or outright impossible to obtain.

Commercial Deep Learning-based ASR systems see widespread deployment, even in languages like Norwegian, where little training data is available. Røyneland et al. 2018 have raised concerns about their (currently lacking) ability to reliably transcribe low-resource languages such as Norwegian in real-life situations and the broader consequences of this in society.

While parallel audio training data is difficult to obtain in large quantities, most languages and domains have massive text corpora available. Recent advances in the field of Natural Language Processing (NLP) are making use of these corpora to pre-train general-purpose Language Models (LMs) to obtain state-of-the-art results in a wide range of NLP tasks with relatively little task-specific fine-tuning (Devlin et al. 2019). In theory, these models should be applicable to ASR as well. However, written text uses language very differently from spontaneous conversations, leaving the question of how such large LMs can improve performance in highly specialized domains when very little transcribed speech is available.

1.2 Goals and research questions

This thesis is part of an ongoing project at Telenor Research to improve ASR for real-life, spontaneous Norwegian conversations. Telenor's ASR system is based on Amodei et al. 2016 and trained on both the publicly available NST dataset ($\approx$ 400h), as well as a small internal dataset of customer service calls ($\approx$ 15h). We are primarily concerned with the latter dataset, which is detailed in section 5.1.1.

This thesis aims to explore how large-scale language models can improve conversational ASR performance in low-resource domains and real-life situations.

Research question 1 How do state-of-the-art LM and integration methods perform in low-resource situations?

As we will discuss further in section 3.2, integration of neural LMs is an ongoing research area, and there is no method consistently outperforming the others. Our primary focus will not be to benchmark the various methods. Instead, we would like to evaluate how such methods perform on spontaneous conversations in a low-resource situation, compared to results reported in the literature.

Research question 2 How can LMs exploit conversational context to improve performance?

Modern language models such as Devlin et al. 2019 are capable of attending over a much larger context than previous LMs. Continuing the work from the specialization project preceding this thesis, we run experiments to determine to what extent LMs capturing conversational context can improve ASR results.

Research question 3 How does the LM training scheme and - data affect results?

One aspect of LM integration rarely discussed in the literature is how to train the LM. As our transcribed speech contains very different language from written text, we hypothesize that the datasets and training scheme used to train the LM will impact the results.

1.3 Contributions

We bring state-of-the-art neural LM integration techniques to two challenging low-resource domains. Using a BERT model, we perform N-best rescoring of beam search outputs to obtain significant performance improvements over the already strong baseline system.

Making conversational context available to BERT in the form of previous utterances is key to our approach. We find that the amount of context required depends on the target domain. In particular, we find that formal parliamentary discussions benefit greatly from increased conversational context. At the same time, too much context can also mislead BERT on Telenor's more unpredictable internal dataset of spontaneous, informal conversations.

Further, we find that fine-tuning procedures play a significant role in BERT's ability to rescore the N-best lists. We propose a data-efficient fine-tuning strategy that uses the baseline ASR system to generate sufficient training examples for BERT, even from a tiny dataset. Fine-tuning BERT this way on a small number of relevant samples performs far better than fine-tuning on a much larger, out-of-domain conversational dataset.

Finally, we analyze the N-best lists and find that beams become relatively homogeneous as the utterance length grows. To remedy this, we propose a diversity term inspired by work in image captioning. However, experiments show that more work is needed for this to become a feasible option.

1.4 Thesis Structure

The next chapter gives a brief theoretic overview of ASR, NLP, and Deep Learning applied to sequential data. Chapter 3 provides a focused review on state-of-the-art methods in NLP. We also discuss how related works integrate LMs into ASR pipelines. In chapter 4, we define our ASR pipeline and explain our LM training strategy. Chapter 5 details the datasets used and explains how the experiments were carried out, while chapter 6 explores the outcome. Finally, chapter 7 summarizes the results in terms of the research questions.

Chapter 2

Background Theory

This chapter gives an overview of the background theory on which this thesis is based.

2.1 Natural Language Processing

Natural Language Processing (NLP) is the study of how human languages can be parsed, processed, and understood by computers. In NLP, the word “document” refers to one unit of text being studied. A document’s size depends on the application and can be anything from a single sentence to a whole book. A corpus is a (usually rather large) collection of documents, i.e. social media posts about a particular topic, newspaper articles, or encyclopedia entries.

2.1.1 Tokens and Vocabularies

Each document consists of a string of tokens. Splitting a document into tokens is called *tokenization*. Very early systems tokenized by splitting on whitespace and punctuation. For this reason, “token” and “word” are sometimes used interchangeably. Smarter, rule-based tokenizer algorithms are available for most active human languages. In addition to splitting at word boundaries, they also separate the word stem from any prefixes and affixes where relevant.

The vocabulary $\mathcal{V}$ is the set of allowed tokens and is often defined as the top k words in the training corpus. Ideally, $\mathcal{V}$ should contain all possible tokens in the language. Even for large training corpora, unseen data will often contain a small proportion of unseen tokens. Thus, the NLP system must be able to handle tokens it has never seen before. This is called the out-of-vocabulary (OOV) problem. It is common to solve this issue at the language modeling level, as discussed in the next section.

A different solution is bottom-up statistical tokenizers such as *WordPiece* (Schuster and Nakajima 2012). In Latin-based languages, these are usually instantiated with a vocabulary equal to the language's alphabet and iteratively add the most common token combinations to the vocabulary. This approach can technically guarantee that all documents using the same alphabet only contain tokens from the vocabulary. What would otherwise be an OOV token is now represented as a previously unseen combination of known tokens (i.e. pieces of words). Deep models (see section 2.3.3) can then infer the overall meaning based on the semantic representations of each token. We will get back to this in chapter 3.

2.2 Language models

A Language Model (LM) assigns a probability to a document, indicating how likely it is to see that document. Formally, an LM operates on documents X of tokens x_i over a (finite) vocabulary $\mathcal{V}$. The LM represents the probability of observing a document, denoted $P(X) = P(x_1, x_2, ..., x_t), x_i \in \mathcal{V}$. $P(X)$ is usually understood to mean $P(X|C)$, where C is a text corpus relevant to the domain or task in question.

Except for toy problems, the set of possible documents over a vocabulary is massive, even for modest vocabulary sizes. In addition, many applications require fairly large vocabularies. Using the popular fastText library (Bojanowski et al. 2017) to estimate order of magnitude, $|\mathcal{V}| = 10^6$. With this vocabulary size, there are $(10^6)^{10!} = (10^6)^{3,628,800} = 10^{21,772,800}$ possible documents if we limit ourselves to documents with less than 10 words. Assigning a probability to each of these documents is impossible. Therefore, all practical LMs are forced to make some simplifying assumptions.

2.2.1 Bag-of-words

The simplest models assume that all tokens in the document are independent, leaving a *bag of words*:

$$P(x_1, ..., x_t) = \Pi_{i=1}^{t} P(x_i) \ .$$

Assuming that $P(x_i)$ is given by the frequency of x_i in the corpus reduces the problem significantly. While this is a very crude approximation, it works very well for certain applications such as information retrieval. Obviously, these models are not suited to generate text since they only suggest the most common words in the training corpus.

2.2.2 n-gram models

Tokens are not independent of each other in most languages, so a natural next step is to include some context from neighboring words. Modifying the assumption from earlier, n-gram models assume that tokens are independent given n neighboring or previous tokens, i.e. a *context* c_i:

$$P(x_1, ..., x_t) = \Pi_{i=1}^{t} P(x_i|c_i), \text{ where } c_i = [x_{i-n}, x_{i-n+1}, ..., x_{i-1}].$$

n-gram models are said to be autoregressive, in the sense that they predict the current word x_i using only previous context $x_{1..(i-1)}$, not making any assumption about future words.

Similar to bag-of-word models, n-gram models are trained by counting. The assumption is now

$$P(x_i|c_i) \approx \frac{\text{occurrences of sequence } [c_i \ x_i]}{\text{occurrences of sequence } c_i} \ .$$

This scales well to large corpora, is reasonably efficient for small values of n, and is therefore used in many ASR systems (including the one used for this project, based on Amodei et al. 2016).

Unseen sequences cause a problem very similar to the out-of-vocabulary problem mentioned above. When encountering an unseen sequence, we set $P(x_i|c_i)$ to a small value. Doing so is called *smoothing*, as it prevents hard rejection of documents containing unseen token sequences. We refer to Chen and Goodman 1999 for a comprehensive review of smoothing techniques. Nevertheless, the probability of encountering an unseen sequence increases dramatically with n. In the extreme case, it is equivalent to the original problem of assigning a probability to every possible sequence.

2.3 Artificial Neural Networks (for sequences)

The LMs described in the previous section are all approximations of the original probability distribution. One can also train a general-purpose statistical function approximator. Given $y = f(x)+\epsilon$, we approximate the unknown function f based on a set of observations (x_i, y_i), $i = 1, ..., m$. ϵ is assumed to be random noise or other unobservable data sampled from a normal distribution $\mathcal{N}(0, \sigma)$. The quality of the approximation is measured by a loss function $\frac{1}{m}\sum_{i=1}^{m} L(y_i, \hat{f}(x_i))$ representing the distance between the observed y_i values and the estimated values $\hat{f}(x_i)$. This is typically referred to as (statistic) supervised learning.

An Artificial Neural Network (NN) is a supervised learning model. NNs are a relatively large field of research, and an in-depth explanation is far beyond the scope of this thesis. For a detailed treatment of neural networks, see Goodfellow, Yoshua Bengio, and Courville 2016. The NN consists of layers of nodes, called neurons. Neuron values in the first layer are set equal to the input example x. Values are then propagated forward through the network by calculating linear combinations and simple non-linear functions of the previous layer's values. Once the forward pass is complete, the last ("output") layer is returned as the prediction $\hat{f}(x)$, and compared to a ground truth y to calculate the error. Then, the network efficiently calculates the error gradient with respect to each parameter and uses gradient descent to minimize the error. This procedure is called *backpropagation*.

Deep networks and parallel optimization

Most aspects of NNs have been subject to extensive research. While most of it is far beyond this text's scope, two findings are worth highlighting.

First, experiments have shown that many layers with fewer neurons each ("deep" networks) usually give better results than few layers with more neurons each ("wide" networks). From a mathematical perspective, a set of parameters should exist where a wide network will approximate f with the same performance as a deep network of the same size. In practice, optimization algorithms struggle to find this set of parameters. Instead, the early layers of the network learn representations of the inputs, which are then used to build increasingly complex abstractions. Intuitively, it is a form of automated feature extraction and - engineering, used by the last layer to predict y.

Second, large NNs tend to perform better as long as the training dataset is large enough to avoid overfitting. In the field of NLP, this is very often the case. With millions or billions of parameters to optimize, we need the backpropagation algorithm to run efficiently. The solution is to represent the problem as matrix calculations and exploit the linear algebra SIMD hardware originally intended for computer graphics (GPUs). GPUs allow for a high degree of parallelization and make it feasible to train large networks, as long as the calculations can be expressed as SIMD operations.

2.3.1 Word Embeddings

Neural networks operate with numeric inputs. Therefore, it is necessary to embed each token or word into a vector. The most straightforward approach is to one-hot encode the tokens, but this forces the model's input size to be equal to the vocabulary size, leaving a very sparse representation. To reduce the dimension size, it is common to use a word embedding model like Bojanowski et al. 2017. Word embeddings use each word's context to (implicitly) capture a word's semantic meaning as a point in vector space.

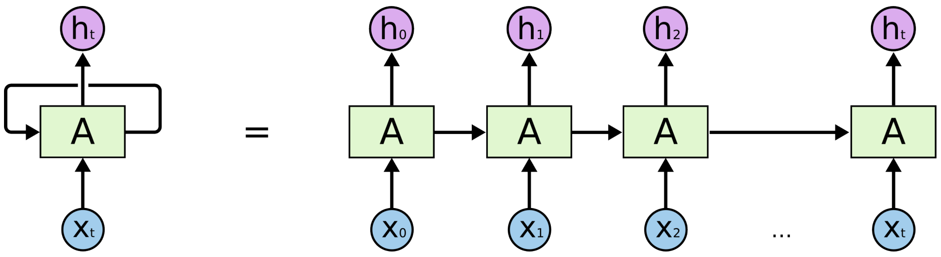


Figure 2.1: Left: A simple RNN. Right: The same network, unrolled for t time steps. Illustration from Olah 2015.

Several ways of building word embeddings have been proposed. One common approach is to use a function approximator with the optimization goal of reducing the distance between co-occurring tokens in a training corpus (Pennington, Socher, and Manning 2014), building on the assumption that words with similar usage also have a similar meaning, as initially hypothesized by Harris 1954. Such approaches produce explicit, reusable word embedding models. Alternatively, embeddings can be trained implicitly as the first layer(s) of a NN.

2.3.2 Recurrent Neural Networks

Sometimes we can exploit information about the problem's structure when designing the NN architecture. This introduction of bias will help the NN better capture the patterns in the data while reducing the number of parameters to optimize. NLP problems are a good example of this. A document can be represented as a matrix using the embedding approach from the previous section. Flattening the matrix into a vector before feeding it to the NN is suboptimal because the sequence structure is lost.

A Recurrent Neural Network (RNN) is a class of NNs exploiting the sequence structure by introducing a time dimension. The sequence is loaded into the network with one vector ("token") at each time step. Each hidden neuron h has a self-connection as shown in figure 2.1, allowing it to use its own value at the previous time step in the calculation of its next value: $h_t = g(x_t, h_{t-1})$. Since each vector passes through the same network sequentially, a smaller number of weights are reused at each time step. Bidirectional dependencies can be captured by adding an RNN module receiving the sequence in the opposite order.

One significant weakness of these basic RNNs is vanishing or exploding gradients in the long backpropagation path from the final outputs back to the first inputs (Y. Bengio, Simard, and Frasconi 1994). Long short-term memory (LSTM) units add a hidden cell state, explicitly written and cleared based on the current

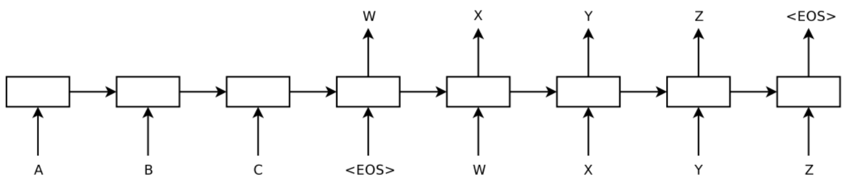


Figure 2.2: A sequence-to-sequence network as proposed by Sutskever, Vinyals, and Le 2014. Note how the entire input sequence must be compressed into the hidden state of the fourth node.

input and previous hidden state. The update equations for this cell state are designed to allow a constant flow of gradients backward through time, thereby solving the vanishing/exploding gradient problem (Hochreiter and Schmidhuber 1997).

2.3.3 Sequence to sequence

A common class of problems is to map one sequence to another, for example machine translation (sequence of strings to sequence of strings) or ASR (sequence of sounds to sequence of strings). A framework for this class of problems is the RNN Encoder-Decoder. An encoder RNN maps the input sequence to a context vector c (usually the last hidden state of the encoder RNN, h_T^{encoder}), which is passed as additional input to the decoder RNN as shown in figure 2.2. The decoder then uses c to generate the target sequence.

Attention

Since c is the only shared element between the encoder and decoder, it must contain all data needed to produce the target sequence. Thus it becomes a bottleneck as the input sequence length grows, limiting the model's ability to "remember" details of long sequences.

Bahdanau, K. Cho, and Yoshua Bengio 2016 propose to have a separate context vector at each time step

$$c_t = \sum_{i=1}^{T} \text{Align}(h_{t-1}^{\text{decoder}}, h_i^{\text{encoder}}) h_i^{\text{encoder}} \ , \tag{2.1}$$

where Align is a probability distribution indicating how relevant the i-th input vector is when decoding the t-th output position. This makes the context vector a weighted sum where the most relevant encoder hidden states at each time step are most prominent.

Later works generalize the concept, borrowing the concepts of queries Q, keys K, and values V from information retrieval. Intuitively, the decoder sends a query to the encoder, which finds the corresponding key (alignment) and returns the associated value. With this notation, equation 2.1 becomes $c_t = \sum_{i=1}^{T} \text{Align}(Q_{t-1}, K_i)\ V_i$.

Align was initially implemented as a NN with a single hidden layer. This approach is called *additive attention*. A simpler option is *dot-product attention*, which computes the attention as a simple dot product:

$$C = \text{Softmax}(Q * K^T) * V\ .$$

The latter approach is far faster but performs worse when the vector sizes grow very large, though this can be at least partially mitigated by scaling the dot product.

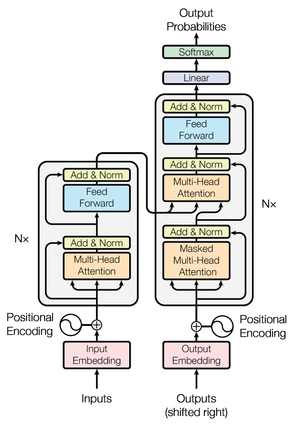
2.3.4 Transformer

Vaswani et al. 2017 show that attention mechanisms can completely replace the recurrent modules. In addition to “traditional” encoder-decoder attention as shown in the previous section, the Transformer uses so-called self-attention layers in the encoder and decoder, replacing the temporal connections in RNN models.

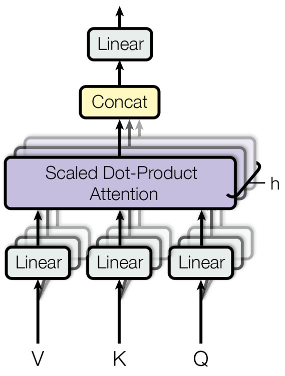
Self-attention is a special case of attention where c_t is computed from a single sequence (this is, $Q = K = V$), for example a specific encoder layer. The layer’s output for each position in the sequence is calculated based on all sequence positions. Compared to the fully connected NN in section 2.3, two key differences are that many of the weights are shared, and the sequence structure is preserved.

Rather than using the full vectors when computing attention, the Transformer computes learned projections of the vectors into several distinct lower-dimensional spaces. After applying scaled dot-product attention on these embeddings, the results are concatenated together and projected back to the original dimension. By computing multiple distinct representations, each attention head embeds different aspects of the sequences. This is shown in figure 2.3b. All parameters of this multi-head attention mechanism are learned through standard backpropagation.

An encoder block in the Transformer consists of two sub-layers: a multi-head self-attention and a feedforward NN. There are residual connections around each sub-layer and a normalization layer to ensure stable training. Each decoder block is mostly identical to an encoder block. The only significant addition is a multi-head attention layer attending to the corresponding encoder’s output. The



(a) Transformer



(b) Multi-Head Attention

Figure 2.3: The transformer architecture as illustrated by Vaswani et al. 2017.

decoder's self-attention layers mask out future positions to preserve the autoregressive property mentioned in section 2.2.2.

Putting it all together, we get the transformer architecture shown in figure 2.3a. The original transformer model (as presented in Vaswani et al. 2017) consists of $N = 6$ encoder blocks followed by the same number of decoder blocks. The input tokens are first passed through an embedding layer learning context-independent word embeddings for each token. Then, a periodic positional encoding is concatenated to the embeddings to maintain positional information.

In this architecture, the recurrent connections are therefore redundant and can be removed. Doing so removes the computational bottleneck caused by the RNN's long backpropagation path. In turn, this allows for a much higher degree of parallelization and faster training. These performance improvements have made many of the Neural Language Models (NLMs) discussed in chapter 3 feasible to train.

While the positional embedding is periodic and the multi-head attention can be computed on any sequence length, Transformer models generally struggle when encountering documents longer than those seen during training. The reason is that since it has never seen such long-term dependencies before, it cannot represent them correctly and therefore underperform (Dai et al. 2019). We refer to this maximum supported document size as the input window size.

2.4 Automatic Speech Recognition

Automatic Speech Recognition (ASR) is the task of recognizing the sequence of words being said in a given audio sequence. Historically, separate models were trained for acoustics, pronunciation, and language and then combined to form an ASR pipeline. Typically, Hidden Markov Models were often used. Getting this to work requires a large amount of domain knowledge and language-specific feature engineering. Deep Learning was first used to replace individual components of the ASR pipeline, such as the phonetic model. As hardware and optimization software improved, deep learning proved capable of replacing practically the entire pipeline with so-called end-to-end models.

In a typical end-to-end ASR system, the input is an audio segment and the target is the transcription of the audio. The audio segment is typically pre-processed into log-mel spectrograms, which provide a numerical representation of the audio signal in the frequency domain, well-known to capture speech features. Details on this pre-processing are beyond the scope of this thesis; we refer to Kamath, J. Liu, and Whitaker 2019 for an introduction. The output representation is typically characters or a small vocabulary built from a bottom-up tokenizer as discussed in section 2.1.1. Using words or large subword vocabularies tends to perform poorly, as the network would need to see many examples of rare words in order to learn their pronunciations (Huang et al. 2019).

The spectrogram representation of the audio is much longer than its transcript since the former is calculated from overlapping windows of $\sim$ 20ms of audio. How much longer cannot be estimated in general because it depends on how fast the speaker was talking, among other factors. End-to-end ASR systems can be categorized by their solution to this alignment issue, as it heavily influences modeling assumptions and many other decisions made when building the system. There are two approaches to solving this: either make the network collect the transcript at the beginning of the output layer or teach the system how to align the tokens.

As this thesis is written in collaboration with Telenor Research, we will use their CTC-based system as our baseline. For this reason, we will also focus more on the "CTC paradigm" in this thesis.

2.4.1 Connectionist Temporal Classification (CTC)

Graves et al. 2006 propose the CTC framework to solve the alignment issue. The approach is to relax the loss function such that the network can choose any sequence alignment as long as the tokens are outputted in the correct order. This way, the network learns to transcribe sequences with reasonably correct alignment.

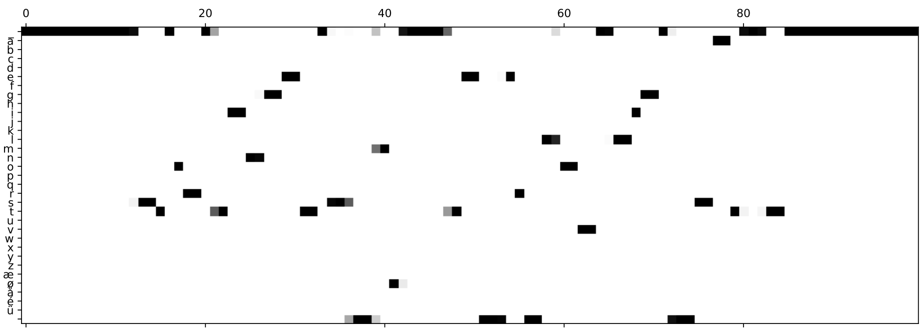


Figure 2.4: Z matrix for the sentence "stortingets møte er lovlig satt". Each column represents the probability of each token (with _ representing `[BLANK]`) at the given time step. Darker color corresponds to higher probabilities. Notice how periods of silence are filled with `[BLANK]`, and the blank token between the double t at the end. Time steps have a duration of 20ms with a 10ms stride.

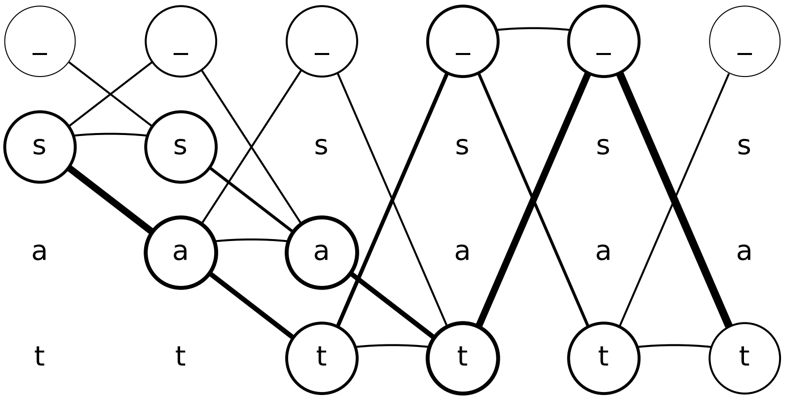


Figure 2.5: The set of paths $\pi \in \text{Paths}(y)$ all collapsing to the string $y =$"satt" when $T = 6$. Thicker lines indicate the number of paths each edge or node is part of. As with figure 2.4, notice how every path must pass through a blank token to produce the double t.

The CTC framework works by allowing the network to output repeated tokens and blanks to fill an output layer $Z \in \mathbb{R}^{|\mathcal{V}'| \times T}$, where T is the number of time steps in the input sequence x and $\mathcal{V}' = \mathcal{V} \cup \texttt{[BLANK]}$ is the vocabulary extended with a special blank token. Each element $z_k^t = P(k|x,t)$ of Z represents the probability of observing a token k at time step t. This output format allows the network to consider any transcript without having a pre-determined alignment between y and x (up to the length of the input audio). An example Z matrix is shown in figure 2.4.

A transcript y is obtained by collapsing one or more alignments, or paths, π through Z. First, repeated tokens are removed, and then blanks are removed to obtain y. A graph visualization of this is shown in figure 2.5. Assuming all time steps are independent given x, we obtain

$$P(y|x) = \sum_{\pi \in \text{Paths}(y)} P(\pi|x) = \sum_{\pi \in \text{Paths}(y)} \prod_{t=1}^{T} z_{\pi_t}^t, \tag{2.2}$$

where Paths(y) is the set of paths that can be collapsed to y and $z_{\pi_t}^t$ is the probability of observing the t-th token of a path π at time t according to Z. The classification can then be expressed as $\text{argmax}_y\, P(y|x)$, and the loss function maximizing the probability of all valid paths[1] is simply $\mathcal{L}_{CTC}(x,y) = -\log P(y|x)$. We refer to Graves et al. 2006 for the proof deriving this optimization objective from maximum likelihood.

Beam search decoder

Evaluating $\text{argmax}_y\, P(y|x)$ during inference is usually infeasible due to the depth and high branching factor. We could assume that the highest-ranked path will correspond with the highest-ranking transcript and find this path using a greedy search in linear time. In some cases, such as the example in figure 2.4 where the AM is very confident, this would give the correct transcript.

In general though, this results in suboptimal transcripts. Equation 2.2 shows that the most likely transcript is the one whose path probabilities *sum* to the highest value. Therefore, we would need to consider multiple paths when looking for the best transcript. CTC-based ASR systems do this using the Prefix Beam Search (BS) algorithm.

Several variations exist; the one we present here closely matches A. Y. Hannun et al. 2014. As with classical beam search, CTC beam search maintains a set Y of the N best prefixes (partial transcripts). Rather than maintaining a single score for each candidate, it maintains the probability of observing each prefix both with

[1]Since z_k^t is a probability distribution over $\mathcal{V}'$, the probability of other paths is implicitly minimized.

and without a [BLANK] token at the end, denoted as $P_b = P(y^{\leq t} \cup \texttt{[BLANK]}\,|x)$ and $P_{nb} = P(y^{\leq t}|x)$ respectively. At each time step $t \leq T$, the algorithm attempts to expand each prefix $y^{<t}$ by appending a token k. It needs to consider three cases:

1. k is different from the last token in $y^{<t}$. In this case, add the new prefix $y^{<t} \cup k$ to Y with probability $P_{nb} = z_k^t(P_b + P_{nb})$ and $P_b = 0$.

2. k is the [BLANK] token. In this case, only update the probability of $y^{<t}$ ending in a blank token: $P_b = z_k^t(P_b + P_{nb})$.

3. k is the same token as the last non-blank token in $y^{<t}$, i.e. a repetition. Similar to above, update the probability of observing this prefix given the new evidence: $P_{nb} = z_k^t P_{nb}$.

Once each time step is calculated, the algorithm prunes all except the N most likely prefixes. Since each transcript's path probabilities are represented implicitly by its two associated probabilities, equation 2.2 allows us to approximate the probability of each partial transcript as

$$P(y_{<t}|x) \approx P_b + P_{nb}. \tag{2.3}$$

At the end of the beam search, we have $\operatorname{argmax}_y P(y|x) \approx \operatorname{argmax}_{y \in Y} P(y|x)$.

Deep Speech

Amodei et al. 2016; A. Hannun et al. 2014 propose Deep Speech, a family of ASR systems based on deep neural networks and the CTC framework. Due to performance concerns when stacking many RNN layers, Deep Speech processes each audio feature only in the context of neighboring features using convolutional layers (see LeCun et al. 1989). Then, it combines them in a single bidirectional RNN layer as shown in figure 2.6. Doing so mitigates the performance loss incurred by the RNN.

Deep Speech 2 builds on the original architecture and, through further optimization, makes it feasible to train deeper models with more data. The NN architecture itself adds more layers of each type[2], but contains practically no other changes. Due to the large model size, curriculum learning is applied, using transcript length as a proxy for difficulty. The remaining optimizations are mostly related to efficient data locality management on the GPUs and parallel execution of the CTC loss function. As these optimizations are particular to their hardware and codebase, we refer to Amodei et al. 2016 for details.

[2]Strictly speaking, Amodei et al. 2016 experiment with many variations of layer counts and types, describing all of them as "Deep Speech". We view such choices as tunable hyperparameters and place little emphasis on exact layer counts and types in this thesis.

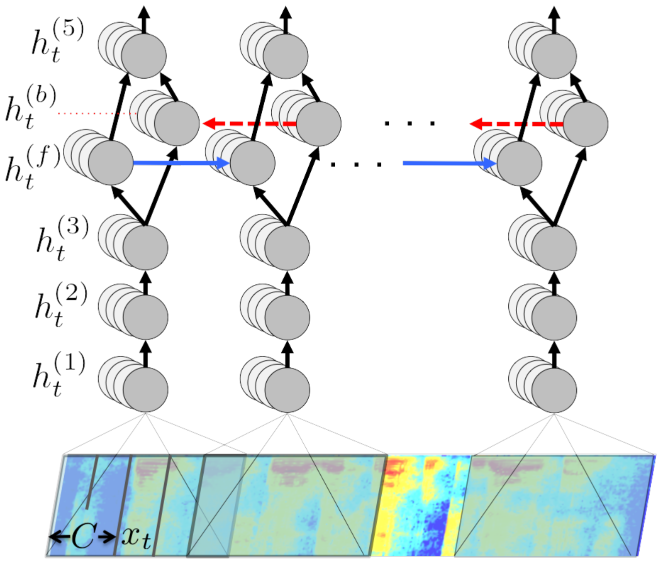


Figure 2.6: The Deep Speech architecture as illustrated by A. Hannun et al. 2014.

Both Deep Speech variants operate at character level. Due to the independence assumption made in equation 2.2, its implicit language model is often found to be lacking compared to sequence-based end-to-end systems. Instead, Deep Speech models rely on an external n-gram LM during decoding (see section 3.2).

2.4.2 The attention-based approach

The encoder-decoder models from section 2.3.3 can also be used for ASR. These solve the alignment issue by encoding the entire audio clip x into an internal representation h and then decoding that representation into y using a generative RNN. These use the attention mechanism to gather information from the correct part(s) of h. Their reliance on this mechanism gives them the name attention-based.

Chan et al. 2016 pioneered this approach, using pyramidal recurrent layers to produce a shorter, more abstract audio representation h. This makes the size of h dependent on the length of x. However, the system still makes all of h available through the attention mechanism at every decoding step, such that no pre-determined alignment is required. Several other variations have been proposed, such as Gulati et al. 2020 proposing to use the Transformer architecture instead of the RNN encoder-decoder.

Compared to CTC, these models do not make the dubious assumption that each character is independent. Therefore, the network more easily learns an internal language representation. Indeed, even the early work of Chan et al. 2016 observed that dictionary constraints were not needed because the model learned to spell correctly. Nevertheless, both sequence-to-sequence and CTC models benefit significantly from an external language model.

For decoding, sequence to sequence models also use beam search. Unlike CTC, classical beam search can be used, but token probabilities are calculated using the RNN decoder. The RNN decoder must be invoked with many different contexts at each time step, making each step more computationally expensive. However, since there is only one path leading to each transcript, the beam size can be made much smaller.

Chapter 3

State of the art

The Transformer architecture introduced in section 2.3.4 was a major turning point in the field of NLP. At first, this was largely due to the strong empirical results from the architecture. Later, Yun et al. 2020 showed that the architecture is a general sequence-to-sequence approximator. A trend lately followed by the NLP community is to build large networks of Transformer blocks and pre-train them for days or weeks on massive text corpora to achieve state-of-the-art performance on different NLP tasks.

3.1 BERT

In order to achieve such impressive performance, Transformer models were believed to build good internal language representations. Devlin et al. 2019 introduced Bidirectional Encoder Representations from Transformers (BERT), a language representation model built directly on top of the Transformer encoder stack. The only significant architectural change was to replace the entire decoder stack with a task-specific output layer. Through self-supervised training, this model learns good language representations, enabling it to perform a range of NLP tasks with minimal task-specific fine-tuning.

A significant advantage of BERT over previous work is that BERT is fully bidirectional. Due to the nature of self-attention described in section 2.3.4, BERT uses the entire input sequence to generate the attention vectors. As a result, the token-level embeddings are updated dynamically based on both previous and future context. This comes at the cost of losing the autoregressive property found in most previous work, though extensions such as Dai et al. 2019 propose workarounds for this limitation.

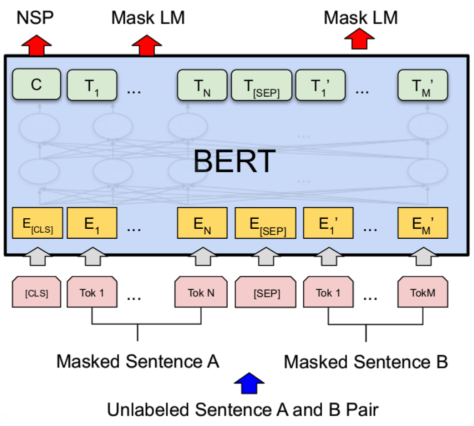


Figure 3.1: Devlin et al. 2019's illustration of BERT's pre-training scheme. A classification token is prepended to the first input sentence, and a separator is placed between them. A position embedding is then concatenated to each token before feeding everything through the model. The first output token corresponds with the NSP task, while the remaining tokens are used for MLM.

3.1.1 Pre-training

BERT is intended to be pre-trained once on a large text corpus and then fine-tuned for each "real-life" task. Pre-training can be done on any unlabelled text corpus but can be considered supervised because the pre-training procedure generates all necessary labels. BERT is pre-trained on two tasks. These are illustrated in figure 3.1 and described below.

Masked Language Modeling (MLM)

Since BERT requires the entire input in its predictions, tasks like guessing the next token from seq2seq models cannot be used. In Masked Language Modeling, also known as Cloze, 15% of the tokens in each input are replaced with a special `[mask]` token or occasionally a random token chosen from the vocabulary. BERT guesses the original tokens, and a loss is computed and backpropagated. Note that loss is only computed for the masked positions to keep the computational complexity manageable.

Next Sentence Prediction (NSP)

Many NLP tasks require understanding more than the contents of a single sentence, but also the relationship between sentences. Devlin et al. 2019 argues this is not captured sufficiently in the MLM task. Instead, they propose Next Sentence Prediction to learn coherence.

NSP is very simple: given two sentences A and B, the model should predict whether B came directly after A in the original text. The training procedure generates samples by setting B to either the sentence following A in the original document (positive sample) or any randomly sampled sentence from a different document (negative sample). The sentences are fed into the model with a special `[sep]` token between them to help the model tell them apart, enabling efficient handling of variable-length sentences.

Note that a "sentence" in this context does not necessarily mean a linguistic sentence. BERT, as implemented by Devlin et al. 2019, does not take natural sentences into account and instead places the split at any whitespace in the text.

Sentence Order Prediction (SOP)

Devlin et al. 2019 claim that both of the previous tasks were necessary, but especially the NSP task has been contested in later work. Y. Liu et al. 2019 found that removing NSP made no difference, raising questions of exactly what BERT learns from the NSP task. Lan et al. 2020 claims that since negative samples come from different documents, the easiest way to solve NSP is by topic prediction since it is already needed for MLM.

Lan et al. 2020 proposed Sentence Order Prediction to force BERT to build some sort of coherence representation. As the name implies, SOP samples two consecutive sentences and asks the model to determine their original order. This way, the model is forced to learn whether the text is coherent. The authors observe that SOP successfully solved the NSP task but not the other way around, strengthening their hypothesis that BERT chooses to solve NSP primarily by topic prediction.

Again, note that SOP as proposed by Lan et al. 2020 operates on text segments rather than linguistic sentences. Thus, while making training more efficient, it also gives the model several obvious coherence cues to work with.

3.1.2 Input/Output representation

Since BERT is a fine-tuning-based approach, inputs and outputs must be reusable for many downstream tasks. The larger the change of IO representation, the more task-specific fine-tuning is needed.

The input text is split on whitespace, and then each word is tokenized with WordPiece (see section 2.1.1). This results in a relatively large vocabulary, as many pre-training corpora contain a very long tail of rarely-used Unicode symbols that must be included to avoid OOV issues. Y. Liu et al. 2019 argue that bytes are just as good symbols as Unicode characters and tokenize the raw text using a byte-level version of WordPiece[1], thereby making better use of the vocabulary space.

The choice of vocabulary is “locked” once pre-training starts, as any change to the vocabulary would require re-training BERT. One could theoretically remove tokens and re-train the embedding layer, but this is compute-intensive and rarely done in practice. Therefore it is important to ensure the vocabulary is not too biased towards any particular domain.

The first input token is always a special `[cls]`, and each sentence ends with the `[sep]` token, which is used actively by BERT during NSP and downstream tasks to tell the sentences apart. The tokenized inputs are then encoded individually into a context-free embedding. BERT appends to this embedding a sinusoidal positional encoding and a segment encoding indicating whether each token is part of sentence A or B.

As mentioned above, the output layer is adjustable depending on the task at hand. The first output position (corresponding to `[cls]`) performs binary classification. During pre-training, it performs NSP, and thus it is suited for document- or segment-level classifications. The remaining output positions are for use with MLM, outputting a probability distribution over the vocabulary. Due to the vocabulary size and sequence length, this can become computationally expensive to compute. If the fine-tuning training set is sufficiently large, the output layer can be adjusted as needed. For example, Devlin et al. 2019 adds two output nodes indicating start and end positions when fine-tuning on the SQuAD dataset. We refer to Rogers, Kovaleva, and Rumshisky 2020 for a review.

3.1.3 Model scaling

A general trend in NLP is towards bigger models and more training data. Brown et al. 2020 showed that Transformer-based LMs are incredibly scalable, to the point where they become too big to evaluate on current hardware. Naturally, this leads to the practical question of how to overcome hardware constraints when scaling Transformer models. Lan et al. 2020 attempts to reduce the number of parameters in BERT to overcome memory limitations, enabling the training of bigger models.

All encoder blocks in the original BERT architecture operate on the same embedding size as calculated at the first input layer. However, increasing the embedding size beyond a certain point gets challenging because the weights needed to calculate the embeddings is a matrix $W \in \mathbb{R}^{|\mathcal{V}| \times H}$, where H is the size of the embeddings. By adding a smaller intermediate layer E, we can decompose this calculation such that we obtain two weight matrices $W_1 \in \mathbb{R}^{|\mathcal{V}| \times E}$

[1]Equivalent to and often referred to as Byte-Pair Encoding. See Shibata et al. 1999 for a discussion in the context of text compression.

and $W_2 \in \mathbb{R}^{E \times H}$ instead.

Most of the parameters in BERT are in the stack of identical Transformer encoder blocks. Inspired by Dehghani et al. 2019, Lan et al. 2020 re-use the same weight matrices for each block. The effect is a substantial decrease in the model size and is argued to function as a form of regularization. BERT can be scaled up much more easily with these two tricks while maintaining high throughput due to better GPU memory locality.

Conditioning

Beyond scaling the model itself, building larger models with longer input windows also mean the model learns to use the entire input window. Since the models are exposed to a range of domains, it will also need a sufficiently long context to determine the type of text in each sample. The most straightforward solution is to fill the input window with similar samples, as mentioned in section 3.9.4 of Brown et al. 2020 when the document itself is too short. However, this solution is not known to generalize to other circumstances.

3.2 Language Models in Speech Recognition

Many state-of-the-art ASR systems use an external LM to improve results. The LM is typically trained separately on a large corpus of unpaired text, separate from the audio transcriptions used to train the AM.

Most systems combine the models at inference time. However, recent advances in neural LMs have enabled deeper integration between the AM and LM when both models are implemented as neural networks. Unfortunately, due to the alignment of outputs in CTC networks, few of these techniques can be incorporated directly into ASR systems built using the CTC framework.

We note that there are few papers comparing the integration techniques. A review by Toshniwal et al. 2018 found that shallow fusion performs better than all other methods tested, but J. Cho et al. 2019 propose refinements and report better results than shallow fusion. Similarly, concurrent research by A. H. Liu, H.-y. Lee, and L.-s. Lee 2019 propose a novel technique and report improvements, but neither experiments have (to our knowledge) been reproduced. Synnaeve et al. 2020 tested many combinations of shallow fusion and rescoring and reported best results when combining both. While we do not intend to benchmark or otherwise compare these methods directly, LM integration is a central research topic of this thesis. Below, we review several integration techniques in more detail.

3.2.1 Inference-time integration

Inference-time integration has been widely adopted in the field for a very long time. Since the models are not combined until they are fully trained, inference-time integration is more general and usually compatible with most model types.

Flexibility in terms of AM and LM types, unfortunately, brings with it some limitations. The main issue is that the integration is typically relatively crude: usually, it boils down to a simple weighted sum of scores from the different models in use. Nevertheless, it is found to work remarkably well in many scenarios.

N-best rescoring

For maximum flexibility in terms of acoustic and language models, it is possible to use the LM to rescore the list of candidate transcripts returned from the beam search. The new score is typically a weighted sum of the AM and LM scores. Unlike the other methods, N-best rescoring cannot bias the AM or BS towards the correct transcript. Therefore, it relies heavily on a good acoustic model and a sufficiently wide beam search.

N-best rescoring works with practically all model types. The external LM only needs to score the final candidate list, making it a good choice when integrating a large LM. As the system needs to wait for the full beam search to complete before rescoring, it is generally not considered streamable.

Shallow fusion

Shallow fusion works by letting a lightweight LM influence which beams are kept and pruned during the beam search. Using heuristics to guide a beam search is a widely adopted trick far beyond ASR and NLP, and shallow fusion is one of the most widely adopted integration techniques. The term shallow fusion was coined by Gulcehre et al. 2015 to contrast it with the neural fusion methods outlined in the next subsection.

Mathematically, it adds a weighted LM term to equation 2.3:

$$P(y_{\leq t}|x) \approx P_b + P_{nb} + \alpha P_{\mathrm{LM}}(y_{\leq t}).$$

This formulation implies that the LM must evaluate every partial transcript considered by the beam search. This subtle detail matters because the LM must handle not only incomplete sentences gracefully but, in many cases, also deal with partial words caused by a character- or subword-level AM. Increasing the beam width can compensate for this, at least to some extent, because it would allow the LM to complete the word before the correct beam is pruned.

Since the LM is invoked thousands of times during decoding, it must be reasonably lightweight. While NLMs can be parallelized efficiently, doing so requires that all strings are scored simultaneously. Beam search implementations, on the other hand, decode distinct *utterances* in parallel. Since utterances are of different length and the threads tend to get out of sync, it becomes challenging to perform efficient SIMD processing (section 2.3) without incurring significant synchronization overhead.

Some variations exist to reduce the number of times the LM needs to be invoked. One example is to combine shallow fusion with a vocabulary constraint. Another trick is to update $P_{\text{LM}}(y_{<t})$ only on word delimiters, caching the previous value in the meantime.

Vocabulary constraints

In many ways, vocabulary constraints are a type of shallow fusion. Like shallow fusion, a simple LM is invoked whenever any token is added to any beam. However, vocabulary constraints are much simpler: they only return a boolean value indicating whether the partial transcript can lead to a string belonging to the language. If the value is false, the beam is pruned from the search immediately. This boolean model can be represented efficiently as a finite state machine, allowing constant-time vocabulary lookups[2].

Effectively, this reduces the search space substantially by simply removing all nonexistent words. The downside is obviously that the ASR system will never be able to output OOV words. We show an extreme example of this failure mode in section 6.4.

3.2.2 Integration by fusing neural models

If both the AM and LM are neural networks, they can share hidden states and representations. Model fusion techniques exploit this property to enable deeper integration between the two models. Several variations of this scheme have been proposed. In this section, we briefly introduce two common approaches.

Deep fusion

Deep fusion is a late integration approach in which the two models are trained separately until convergence before fusing them. Fusion is done by feeding the AM decoder's hidden state and previously predicted tokens as input to the LM. Then, the LM and AM outputs are combined in a final hidden layer. The integration introduces several new parameters to the model, which are optimized through standard backpropagation. AM and LM parameters are usually frozen at this stage, keeping the cost of computing the fusion layer at a minimum.

[2]Because each lookup operation simply attempts to advance the FSM one step.

Cold fusion

Unlike deep fusion, cold fusion is an early integration approach, where a pre-trained LM is fused onto an untrained AM. Then, the AM is trained while keeping the LM parameters fixed. Modeling-wise, cold fusion is very similar to deep fusion. We refer to J. Cho et al. 2019 for a treatment of modeling variations.

3.2.3 Knowledge transfer

Finally, the LM's *knowledge* can be integrated into the ASR system during training. This removes the need to run an LM at inference time, at the cost of making training more complicated. A wide variety of methods have been proposed; a few examples include:

- Multitask learning. The training objective alternates between the standard ASR objective (for example, CTC) and an LM objective (Toshniwal et al. 2018).
- Adversarial training. The AM is trained as a generator to "fool" a criticizing LM (A. H. Liu, H.-y. Lee, and L.-s. Lee 2019).
- Distillation. Rather than using the ground truth transcript as the optimization goal, train the ASR system to output the same probability distribution as an LM (Futami et al. 2020).

3.3 Diverse Beam Search

The size of the search space grows exponentially with the number of time steps ($|\mathcal{V}|^T$), but the part of the search space explored by the beam search grows linearly ($T \cdot N$), and the part being returned is constant (N). In terms of the LM integration methods in the previous section, this means that neural fusion methods expose practically the entire search space to the LM, shallow fusion exposes $\frac{T \cdot N}{|\mathcal{V}|^T}$ of the search space, while N-best rescoring only exposes $\frac{N}{|\mathcal{V}|^T}$. This highlights a problem with N-best rescoring: for it to work effectively, the N transcripts it is exposed to must be an interesting subset of the search space.

Therefore, it is relevant to understand what gets returned from the beam search. At each time step t, the lowest-scoring beams are pruned from the search. Since the non-pruned beams, by definition, has the highest probabilities, any extension of those beams is also highly likely to have a high probability. If this were not the case, the search would quickly stagnate and return incomplete

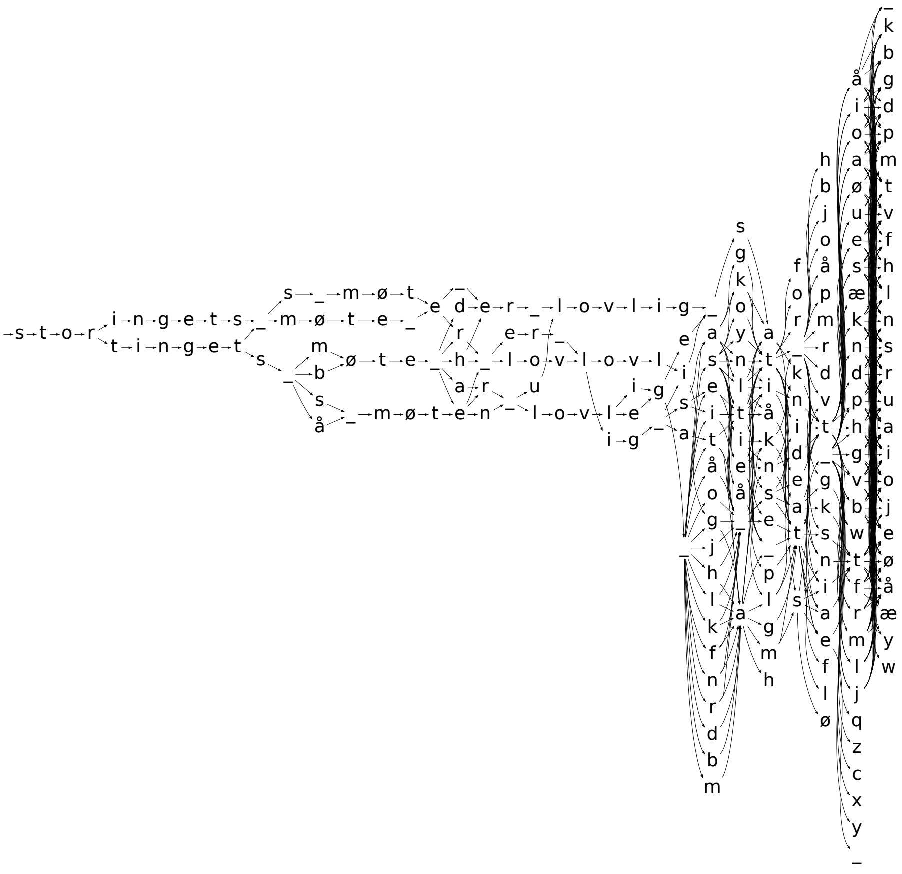


Figure 3.2: A compressed graph visualization of the N-best list from decoding the example “stortingets møte er lovlig satt” from figure 2.4. While there are hundreds of variations among the last 6-7 characters, barely any variations closer to the root are retained.

transcripts[3]. Over time, this accumulates such that early variations are more likely to be pruned.

Eventually, the search terminates, returning the N highest-ranking beams. Since low-scoring beams have been pruned throughout the search, the resulting candidate list frequently only differs in the last few words. An example of this failure mode is shown in figure 3.2. For applications where only the top transcript is used, this is not a problem. However, it severely limits N-best rescoring, as it would be less able to fix mistakes in the early part of the transcript.

This has received relatively little attention in the literature, but some techniques exist to increase diversity. Vijayakumar et al. 2018 propose partitioning the search into groups $g \in [1..G]$ of size $M = N/G$. Each group only considers extensions from within the same group $\mathcal{Y}_t^g = \{y_{t-1} \in Y_{t-1}^g \wedge y_t \in \mathcal{V}\}$. The search adds a diversity bonus to beams that are dissimilar to previous groups $Y_t^{<g}$ according to a distance measure Δ,

$$Y_t^g = \underset{[y_1^g,\ldots,y_M^g] \subset \mathcal{Y}_t^g}{\operatorname{argmax}} \sum_{m=1}^{M} \Theta_t\left(y_m^g\right) + \lambda \sum_{h=1}^{g-1} \Delta\left(y_{m,t}^g, Y_t^h\right) \ ,$$

where $\Theta_t(\cdot)$ is the scoring function used in standard beam search.

Intuitively, each group explores one *mode* of the output distribution. There is no intra-group bonus, as multiple beams are often needed to find the best instance of each mode. The trade-off between thoroughness and regions covered can be controlled by tuning λ and the group count G.

The most common class of distance measures simply average the distances between the beam and each group member:

$$\Delta(a, B) = \frac{1}{|B|} \sum_{b \in B} \delta(a, b) \ .$$

Vijayakumar et al. 2018 propose several implementations of δ, including normalized Hamming distance, n-gram diversity and neural sentence embeddings such as Pennington, Socher, and Manning 2014. The Hamming distance encourages the search to select a different token from the other group at each time step, but can be circumvented by even a single-token alignment change. On the other hand, n-gram diversity penalizes reusing the same n-grams regardless of position. As a completely different notion of similarity, neural sentence embeddings encourage semantically distinct beams. Vijayakumar et al. 2018 report comparable performance with all three diversity measures.

[3]And when it does, it is common to introduce an artificial word count bonus to boost the probability of longer beams.

Chapter 4

Methodology

We base our Speech Recognition system on Telenor's preexisting pipeline, as its components are already tuned to perform optimally on spontaneous conversations. This pipeline is adapted from the Deep Speech 2 architecture, which we extend to perform N-best rescoring with BERT.

Despite the "end-to-end" claims made about Deep Speech and many similar systems, we nevertheless find it beneficial to decompose the architecture into three distinct components: Acoustic Model (AM), Beam Search (BS), and Language Models (LMs). A diagram of the complete system is shown in figure 4.1.

The AM takes an audio segment x as input and outputs a matrix Z such that each element z_k^t represents the probability of token k at time t. The BS decodes Z into a set Y of the N best candidate transcripts, aided by an n-gram LM through shallow fusion. We extend this architecture to perform N-best rescoring of $y \in Y$ with BERT to obtain the final ranking. In addition, we extend the BS decoder by adapting the diversity term introduced in section 3.3. The following sections describe each component in more detail, as well as how we train BERT.

4.1 Acoustic Model

The Deep Speech system is built on the CTC paradigm. This makes for a relatively straightforward neural network architecture with log-mel spectrograms as input and character-level outputs. Telenor Research has made some changes related to signal preprocessing and data augmentation to improve results with the phone call data described in section 5.1.1. For the effects of this thesis, however, it is equivalent to the original Deep Speech 2 model described in section 2.4.1.

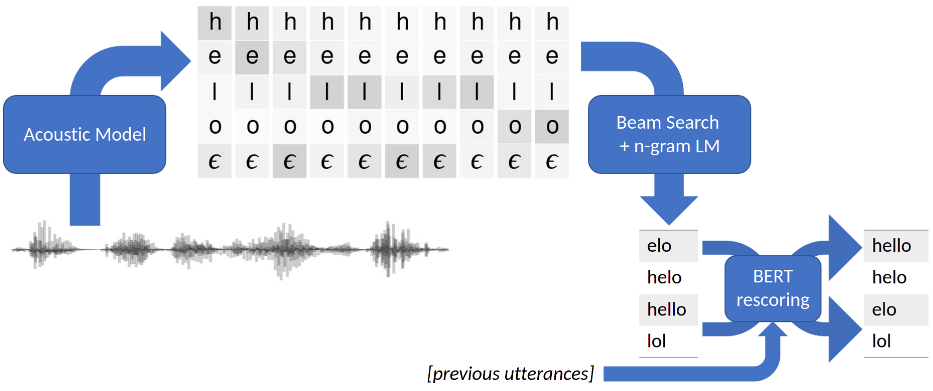


Figure 4.1: Diagram of the ASR pipeline. Audio is processed through an Acoustic Model to produce the matrix Z. The Beam Search produces an N-best list of candidates from Z, guided by an n-gram Language Model through shallow fusion. Finally, the N-best list is then rescored by BERT, taking previous utterances from the conversation into account in order to disambiguate the candidates better.

4.2 Beam Search

As in most CTC-based systems, our AM struggles to learn an internal LM and relies on a good decoding strategy. We use the CTC beam search decoder described in section 2.4.1 to search for the transcript y maximizing

$$P_{\mathrm{BS}}(y|x) = P_{\mathrm{AM}}(y|x) + \alpha\, P_{\mathrm{LM1}}(y) + \beta\, \mathrm{WC}(y)\ , \tag{4.1}$$

where P_{AM} is the AM score sum for the transcript as estimated by equation 2.3, P_{LM1} is the transcript's probability according to an n-gram LM, and WC is a word count bonus to encourage the system to output more tokens even though this would reduce P_{AM}.

We implement LM1 as a Kneser-Ney n-gram model. Rather than training it on external data, we observe near identical performance when training it only on the transcripts used to train the acoustic model. We thus hypothesize that for domain-specific datasets, relevance compensates for abundance. Similar to A. Hannun et al. 2014, we need to increase the beam width to work around the issues with incomplete words as discussed in section 3.2.1.

We denote the highest-ranked transcript from the beam search as

$$y_1 = \operatorname*{argmax}_{y\in Y} P_{BS}(y|x). \tag{4.2}$$

For the baseline system, y_1 is returned as the final transcript, and its WER compared to the ground truth y_{gt} is used to evaluate the system's transcription quality.

4.2.1 Diversity

As discussed in section 3.3, beam searches often struggle with a lack of diversity when processing long sequences. To remedy this, we adapt the Diverse Beam Search to work within the CTC framework. Since our beam search implementation opereates on transcript level (representing paths only implicitly), it is straightforward to add the diversity term to each partial transcript. The set of beams to keep from each group g at each time step will be

$$Y_t^g = \underset{[y_1^g,\ldots,y_M^g]\subset\mathcal{Y}_t^g}{\operatorname{argmax}} \sum_{m=1}^{M} \left[P_{BS}(y_m^g|x) + \lambda \sum_{h=1}^{g-1} \Delta\left(y_m^g, Y_t^h\right) \right], \tag{4.3}$$

where P_{BS} is the probability assigned by standard beam search given in equation 4.1.

Due to the width of a typical CTC beam search, the choice of distance measure Δ has a substantially higher performance impact compared to the experiments of Vijayakumar et al. 2018. For this reason, we stick with the simplest distance measure: character-level Hamming distance. Since many beams follow the exact same path, a Hamming distance should give a sufficient bonus to any beam deviating from this.

4.3 N-best Rescoring

Section 3.2 describes various ways to integrate language models. Several of them require significant engineering effort to implement, despite yielding minimal or no performance gains compared to the other options. Deep and cold fusion, for example, require replacing the entire ASR pipeline with a seq2seq model. Shallow fusion with a large-scale LM like BERT would work but is very computationally expensive. As it is uncertain whether any of the methods can improve results over the baseline, and given the time constraints, we instead choose the more straightforward solution: N-best rescoring.

Although many variations of N-best rescoring exist, we use a straightforward form where we interpolate the scores from the beam search and the secondary language model (BERT):

$$P_{\text{rescored}}(y|x) = (1-\gamma)P_{BS}(y|x) + \gamma P_{\text{LM2}}(y). \tag{4.4}$$

The top candidate after rescoring is simply

$$y' = \underset{y \in Y}{\operatorname{argmax}} P_{\text{rescored}}(y|x).$$

Despite its simplicity, N-best rescoring is a good fit with our existing Deep Speech-based system: it requires few changes to the existing pipeline, and it makes use of the many candidate transcripts returned from our wide beam search. Being decoupled from the AM training also makes it far easier to measure the impact of introducing the external LM by reusing the same AM across all experiments.

N-best rescoring relies on the assumption that y_1 in equation 4.2 is often a suboptimal choice of transcript from Y. However, it is also bounded by the best and worst candidates in Y. To identify the upper performance bound, we can define an oracle rescoring algorithm that always selects the best transcript from those present in Y:

$$y_o = \underset{y \in Y}{\operatorname{argmin}} \operatorname{WED}(y_{\text{gt}}, y), \tag{4.5}$$

where WED is the word-level edit distance between two transcripts.

As will be clear in the results chapter, the edit distance between the best and worst candidates is disproportionately small for long utterances. The reason is that the ratio between potential transcripts to consider and N grows quickly with utterance length. For shorter utterances, the opposite effect occurs, and the search ends up returning a larger portion of the search space. If parts of the search space are disallowed due to vocabulary constraints, the BS may return the entire (legal) search space as Y for the shortest utterances. In this case $y_{\text{gt}} \in Y$ unless the ground truth contains OOV tokens[1].

In practice, since we are transcribing spontaneous conversations, it would be tough for a language model, or even a human, to correctly identify y_o without access to the audio. After all, the candidate transcripts and contexts alone often do not contain sufficient information to determine what was said, and even with the audio available, there are often multiple correct interpretations.

4.4 BERT for spoken language

We use BERT as described by Devlin et al. 2019 for our neural language model. A challenge in using BERT is that transcripts of spontaneous conversations read very differently from, for example, a novel. People tend to use a very different sentence structure when speaking (such as incomplete sentences and repetitions)

[1]This is also to a large extent why the oracle WER for short utterances in figure 6.2 is nonzero.

and a different choice of words, including fillers/hesitations. When two people are communicating, there are also many cases of overlapping speech and interruptions (of both themselves and the other person). In Appendix A we show how the language representations built when training on written text could be suboptimal for inference on conversations and vice-versa. To deal with this, BERT needs to be fine-tuned to the task at hand. It also needs to learn spoken language.

We propose three different strategies for training BERT, as explained in the following sections.

4.4.1 MLM & NSP

The simplest solution is to keep training BERT with the same objective used during pre-training. These are known to teach BERT a good language representation, and previous work such as Shin, Y. Lee, and Jung 2019 report performance improvements over the baseline. Using the NSP head for scoring is straightforward, as it already outputs the probability directly. MLM, on the other hand, is more complicated. With the MLM output nodes, we can score sentences as follows:

1. Pass the sequence $y_{1..T}$ through the model, masking the first position.
2. Among the output probabilities, note the probability of the token at the first position being the original token, denoted $P(y_1|y_{2..T})$.
3. Repeat for each remaining position in the sequence.
4. Calculate the product $P(y) = \prod_{t=1}^{T} P(y_t|y_i \forall i \neq t)$

This approach requires T forward passes through the network (Shin, Y. Lee, and Jung 2019). Using GPUs and similar SIMD hardware, scoring a single utterance can be done in a single batch. However, the memory requirement to store the result will be $\Theta(T^2|\mathcal{V}|)$. As shown in Appendix B, MLM scoring is overly computationally expensive for rescoring thousands of beams, let alone integrate in a CTC beam search.

4.4.2 Conversational NSP

In NSP as proposed by Devlin et al. 2019, BERT's input is fully packed with text, with the separator being placed anywhere inside the input text. This reduces BERT's ability to make use of the conversational aspect of our domain. To remedy this, we adapt the concept of NSP to conversations: positive samples are simply triplets of consecutive utterances, while negative samples are generated by replacing the third utterance with a different one.

Replacements are sampled from other conversations in the same way as Devlin et al. 2019. Unlike Y. Liu et al. 2019 and Lan et al. 2020, preliminary experiments indicate that sampling from the same conversation does not improve performance. We attribute this to differences between written and spoken language: written text is always linear, while overlapping speech is common, making the sentence order ambiguous.

4.4.3 Disambiguation task

With the training tasks proposed so far, there is an apparent train-test mismatch: the N-best lists primarily contain variations of the same text, often with only a few words differing. Spelling - and grammatical errors are also far more common, neither of which are captured by the preceding tasks.

To remedy this, we propose a straightforward disambiguation task for training NLMs. For BERT-like models, this task retrains the NSP classification head, denoted `[CLS]` in figure 3.1. Assuming a dataset $\mathcal{D}$ of candidate transcripts Y and ground-truth conversational context c, the disambiguation task optimizes

$$\min_{\theta} \sum_{(c,Y)\in\mathcal{D}} \left[\mathcal{L}\left(1, P_{LM_\theta}(y^*|c)\right) + \sum_{y\neq y^*} \mathcal{L}\left(0, P_{LM_\theta}(y|c)\right) \right]$$

where $\mathcal{L}$ is the cross-entropy loss function and y^* is either the ground truth y_{gt} or y_o (equation 4.5). As we will also show empirically in table 6.4, using y_o as the target is more reliable than y_{gt}. For "easy" samples, $y_o = y_{\text{gt}}$, making the distinction irrelevant. For more difficult samples however, y_{gt} is very distinct from the predicted transcripts in Y. This makes the task rather easy, not preparing BERT for reality at inference time.

The sum in the second term is over the remaining transcripts in Y. Because summing over all other samples can give a highly imbalanced dataset, we artificially restrict the number of samples processed. We found this to yield better model performance in preliminary tests and, except where stated otherwise, randomly sample only one or two transcripts with strictly higher WER than y_o.

Though we did not find this necessary in our experiments, the training data can be augmented by adjusting the parameters of the beam search or running a training epoch on the AM. Such simple augmentation will slightly randomize the results and should generate a sufficient number of distinct examples to compensate for the lack of training data.

4.4.4 Input representation in BERT

Rogers, Kovaleva, and Rumshisky 2020 review several studies showing how BERT learns to rely on the input representation being used during pre-training. Due to the small size of our conversational datasets, we choose to retain the same input structure as described in section 3.1.2:

$$\text{[CLS]}\ c_{-2}\, c_{-1}\ \text{[SEP]}\ y\ \text{[SEP]}$$

Devlin et al. 2019 also append segment embeddings in addition to the positional embeddings. We reuse these to indicate whether each token belongs to the context or the current utterance. The context of samples exceeding BERT's input window size is truncated. Due to batch processing, shorter candidates are padded with `[PAD]` tokens.

4.4.5 Limitations

There are several limitations to our use of BERT. In particular, we are forcing BERT to read all conversations linearly. As a result, all information about overlapping speech is being stripped. Xiong et al. 2017, for example, train an LM capable of understanding overlapping speech. While we do have speaker tags available and could obtain approximate segment alignments, we consider it unlikely that BERT will be able to relearn new meanings of the segment embeddings in our low-resource domain.

Further, we leave much of BERT's input window empty. While BERT generally tends to work better with more context, it could also prevent BERT from picking up topic changes in our spontaneous conversations. In addition, by training BERT to rely on very long conversational context, we could prevent it from performing well at the beginning of a conversation. Furthermore, the entire system becomes more computationally expensive to run.

Chapter 5

Experiments

We test several variations of shallow fusion with n-gram language models, rescoring with BERT, and the effect of introducing a diversity term in the beam search. First, we train a baseline acoustic model with a very simple greedy inference strategy. Next, we replace the greedy inference with beam search. The beam search is repeated with and without n-gram fusion. Then, we rescore the N-best lists from each beam search output using several BERT models. All experiments are then repeated with the diversity term in place. For each combination, we record the WER as well as other metrics and qualitative samples as presented in chapter 6.

This chapter explores the practical aspects of our experiments. First, we describe the datasets and the preprocessing we applied to them. The following sections then explain details of the various components of the system. Finally, we note some technical details regarding hardware and implementation concerns. Due to the complexity of the ASR pipeline, documenting every single implementation decision in prose is unrealistic. Consequently, we publish our source code[1] both to make our results reproducible and to encourage further research.

5.1 Datasets

This experiment involves five data sources: text from the Norwegian National Library and OpenSubtitles, and speech with transcriptions from Nordisk Språkteknologi, Telenor Norway's Customer Service, and the Norwegian Parliamentary Speech Corpus.

[1] `https://gitlab.com/sburud/master`

5.1.1 Telenor Norway's Customer Service (TNCS)

Telenor Research transcribed 150 phone calls recorded by Telenor Norway's customer service[2], totaling approximately 15 hours of audio. These conversations were spontaneous and transcribed verbatim. Transcriptions are separated into customer and agent channels and split on silence into "utterances".

Two utterances of a typical conversation are shown in table 5.1. The customer speaks a dialect that maps closely to Nynorsk, so the resulting transcript is a mix of Bokmål and Nynorsk. It illustrates several distinct features of this dataset compared to many publicly available ASR datasets:

- Ungrammatical and incomplete sentences and interruptions.
- Topic changes without clear discourse markers.
- Hesitations and other filler words.
- A wide range of dialects, both in terms of pronunciation and vocabulary.

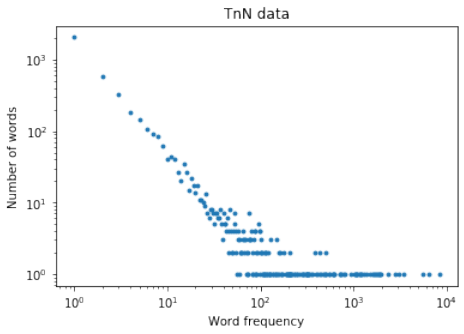
The word frequency distribution is shown in figure 5.1a. A large portion of the distinct words are customer names, addresses, and telecommunication jargon. For the experiments, we remove all hesitations ("uhh" and so on) and standardize the spelling of common words with many spelling variations (for example "okey/okei/ok" and some typical cases of Nynorsk words being very similar to their Bokmål equivalents) to reduce the number of word variations the LMs need to learn.

After preprocessing, the dataset consists of 143 conversations, totaling 10k utterances and 115k words after preprocessing. We use approximately 80% of the utterances for training and split the rest equally into validation and testing sets. The splits are by conversation such that no conversation is spread across different splits.

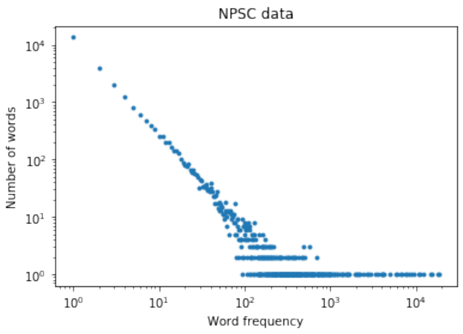
Agent	eee men det det sim kortet der skal hvert fall brukes i den ruteren der da så får du det andre sim kortet nå snart så
Customer	ja eg veit ikkje eg har ikkje fått montert det enda eg fikk den første i dag og det står fri montering kven som monterer det vet du det

Table 5.1: Two consecutive utterances of a conversation before standardizing spelling variations and removing hesitations.

[2]This data and its sharing are regulated by customer consent and confidentiality agreements.



(a) TNCS



(b) NPSC

Figure 5.1: Word frequencies and number of words with each frequency.

5.1.2 Norwegian Parliamentary Speech Corpus (NPSC)

Official political meetings at the Norwegian Parliament (Stortinget) are recorded and published online along with professionally proofread proceedings. The meetings typically consist of pre-written speeches and statements, along with moderated debates afterward. This is also reflected in the language, which tends to be very formal. Since MPs come from all areas of the country, a wide range of dialects are used. Proceedings consist of a mix of Bokmål and Nynorsk, depending on the dialect of each speaker.

While official proceedings are available, these are intended for reading and tend to deviate from the exact words being said. Using ASR, significant preprocessing, and manual review, the National Library adapts the proceedings to match the actual audio. This work is published as the Norwegian Parliamentary Speech Corpus (NPSC)[3] and contains 58 hours of audio. As shown in figure 5.1b, the word frequency distribution is comparable to the TNCS dataset. Below, we show an example utterance after preprocessing:

> i denne perioden kan det og søkes om investeringstilskudd til renovering og utskifting av eksisterende bygningsmasse

Similar to TNCS data, we split the dataset by conversation, treating each day of meetings as one long conversation. We reserve one session for evaluation and one for testing, leaving the rest as training data.

[3] `https://www.nb.no/sbfil/talegjenkjenning/npsc/v0_1/NPSC_01_doc.pdf`

5.1.3 Nordisk Språkteknologi (NST)

Nordisk Språkteknologi (NST) collected large speech datasets in the early 2000s before their bankruptcy in 2003. This data was then purchased by a group of universities and eventually made public[4]. All speech is pre-planned and recorded in a noise-free environment with carefully documented equipment and procedures. The corpus being read is designed to be phonetically balanced, and speakers were sampled to ensure a reasonably balanced set of dialects, genders, and ages are represented. On average, each audio sample is approximately 5 seconds long, and for all practical purposes, each sample is independent of the others. In total, the Norwegian part of the dataset consists of 395 hours of audio.

5.1.4 Colossal Norwegian Corpus

The National Library of Norway has collected an internal "Colossal Norwegian Corpus"[5]. It consists of high-quality Optical Character Recognition (OCR) scans from approximately half the books published in Bokmål and Nynorsk in the past 200 years, as well as several newspapers, government documents, and crawled websites. After deduplication, the corpus consists of around 18B words. As described in section 5.4.2, this dataset is used to pre-train BERT.

5.1.5 OpenSubtitles

The OpenSubtitles Norwegian v2018 dataset consists of the subtitle files for 14,225 movies and TV episodes, along with metadata such as title, year, and genre. A subtitle file is a set of strings with associated timestamps indicating when they are presented on the screen. Lison and Tiedemann 2016 describes them in more detail.

As is expected from a crowd-sourced repository, the quality of some subtitle files is subpar. The most prevalent issues are malformed timestamps and incorrect OCR scans of old movie subtitles. We have ensured almost all strings appear in their original order and removed conversation turn marks, but otherwise left the data as-is.

A movie typically consists of many conversations. We treat each conversation as a document and treat each line within the document as an utterance. Without the actual movies available, we need to make assumptions about where conversation splits are solely based on the subtitle files. While this could theoretically be solvable with NSP, we settled with a much simpler solution: consider a conversation to be ended after t seconds of silence, with $t = 4$ being the best value found empirically.

[4]https://www.nb.no/sprakbanken/ressurskatalog/oai-nb-no-sbr-13

[5]https://github.com/NBAiLab/notram/blob/debd6a/guides/corpus_description.md

5.2 Evaluation

The overall goal is to improve transcription quality on the NPSC and TNCS datasets. As a proxy for transcription quality, we use the Word Error Rate (WER). WER is a normalized word-level edit distance (WED) between two strings, i.e. the number of substitutions, insertions, and deletions needed to transform the predicted transcript into the target transcript, divided by the number of words in the target transcript.

A subtle but important detail is that we report the total WER. The total WER is the total edit distance for all samples being evaluated, divided by the total number of words. Unlike the averaged utterance-level WER, total WER penalizes every mistake equally, while average WER places higher penalties on errors in short transcripts. Many researchers do not specify this explicitly, in part because the difference does not matter in practice. However, on the TNCS dataset, a large portion of the transcripts are very short, making this distinction important.

For N-best rescoring, there is an upper bound on the possible performance improvements, as discussed in equation 4.3. While the oracle rescorer is, to a large extent, an impossible goal given its reliance on the target transcript, it often makes sense to report rescoring improvements with respect to this gold standard. Following Ma and Schwartz 2008, we use the WER recovery rate (WERR) to measure this. In terms of N-best rescoring as described in section 4.3, it is defined as $WERR = \frac{WER(y_1) - WER(y')}{WER(y_1) - WER(y_o)}$, where y' is the rescored transcript.

5.3 Acoustic Model

Our acoustic model (shown in figure 5.2) is relatively straightforward: three strided convolution layers with large kernels and clipped tanh activations, followed by nine 1200-dimensional, bidirectional GRU layers and a single fully-connected layer with the softmaxed character outputs (Z) at the end. Batch normalization is applied between each layer.

The model is trained using the CTC loss function, optimized using stochastic gradient descent with Nesterov momentum and L2 regularization. We first train the model until convergence on the NST dataset, then “fine-tune” on TNCS or NPSC until the early stopping criterion is met on an evaluation set. Telenor Research has already put significant work into tuning the acoustic model, and we do not perform any further hyperparameter tuning of the acoustic model.

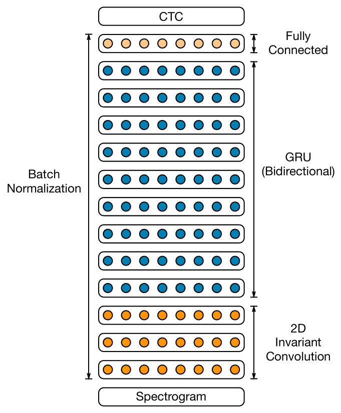


Figure 5.2: The AM architecture. Figure adapted from Amodei et al. 2016.

5.4 Language Models

As outlined in chapter 4, the ASR pipeline will use two LMs: an n-gram model for shallow fusion and a BERT model for rescoring. This section details the training and implementation of these LMs.

5.4.1 n-gram

We train n-gram models of order 2 and 6 using the implementation of Kneser-Ney smoothed n-gram estimation from Heafield et al. 2013[6]. We train separate models for each speech corpus and prune unique n-grams of order 2 and above. All models are trained on the same conversations as the acoustic model. Each utterance is treated as a separate document, meaning no context information from previous utterances is available in the n-gram models.

5.4.2 BERT

Even though Google's original BERT implementation is publicly available, we use a reimplementation by Wolf et al. 2019. Unlike the original, this reimplementation is fully open and is independently verified to reproduce the results of Devlin et al. 2019 from scratch. More importantly, it is more modular and thus easier to adapt to our needs.

Our BERT model follows the structure of "BERT cased multilingual". It is

[6]Specifically, version `35835f1` from `https://github.com/kpu/kenlm`

pre-trained by the National Library AI Lab[7] from an unspecified checkpoint of Google's multilingual model. As this model comes with an extensive vocabulary covering many glyphs never encountered in Norwegian speech recognition, the model is more computationally expensive than necessary. For this reason, we do not report model runtimes. Instead, we leave for future work the task of pruning the vocabulary for improved efficiency and performance.

For our fine-tuning, we use a batch size of 768 and otherwise apply the hyperparameters listed under $\text{BERT}_{\text{base}}$ in Devlin et al. 2019. When we list a BERT model trained on multiple datasets, this is to be understood as sequential training: the model is first trained until convergence on the first dataset, then trained until convergence on the next dataset. Due to the large differences in size, training on multiple datasets simultaneously would prevent contributions from the smaller datasets. Mixing data from different domains during training could also make the learning task harder and negatively impact performance.

When training for disambiguation, we obtain y_o by using the inference results from the AM on training data, which hypothetically could lead to a train-test mismatch. In practice, it means BERT will consistently give slightly higher scores than if we trained on y_{gt}, but those scores get scaled back down by reducing γ in the hyperparameter search.

For simplicity, we always assume that previous utterances were decoded perfectly. This assumption makes the rescoring experimental setup far simpler since all samples can be processed independently and in parallel, similar to standard machine learning pipelines. Theoretically, this introduces another slight train-test mismatch. As discussed previously, BERT works better when the language use is consistent throughout the document, which might not be the case if we feed the ground truth context and the current utterance failed to decode. Further, BERT attends more to the overall content of the context rather than the exact words (see section 6.4), meaning that a few incorrect words in the context are unlikely to affect results. Based on this, we consider it unlikely that this could artificially improve results: if anything, the language mismatch between context and current utterance could make the task slightly harder.

5.5 Beam Search

We use a highly optimized, data-parallel beam search implementation[8]. For memory efficiency, the search is represented as a prefix tree. The N highest-ranking nodes are kept in the beam search ("beam nodes"), and all other nodes

[7]The exact model is available at `https://huggingface.co/NbAiLab/nb-bert-base/tree/6aad23f6c2ed0df8498397175ec07d497f6319a1`

[8]`https://github.com/parlance/ctcdecode`

without beam node descendants are pruned. Technically, this allows a non-beam-node may be promoted to a beam node again after falling out of the search.

At every time step, we consider adding every possible character to every beam node, except for characters only leading to OOV words. If the new character is non-blank, we add the n-gram, word count, and diversity bonuses.

The vocabulary constraint means that our system cannot output OOV words under any circumstances. Instead, it will attempt to assemble the relevant sequence of characters to the closest in-vocabulary interpretation. Never considering OOV words may seem overly strict, but on the other hand, it will primarily filter out garbage words and uncommon proper names. This behavior is also true to the original Deep Speech implementations. We will show examples of cases where the vocabulary constraint both degrades and improves results in section 6.4.

In the diverse beam search, we maintain G completely separate prefix trees. Diversity bonus is only given to beams selecting different tokens at approximately the same position. We use a maximum length difference of 10 characters.

5.6 Hyperparameter tuning

Like many machine learning systems, our system has many hyperparameters to tune. Theoretically, it would be best to tune all hyperparameters simultaneously, but the resulting search space turned out to be far more extensive than our compute budget, even when applying aggressive intermediate result caching. Instead, we perform hyperparameter tuning in two stages: first the beam search parameters, then the rescoring interpolation parameter. The two-step search converges with much fewer trials, and each trial runs faster.

For the beam search parameters, we perform a random search using Optuna (Akiba et al. 2019). We perform a total of three variations of the Optuna search:

- Tune α and β, minimizing $\sum \text{WED}(y_1, y_{\text{gt}})$. The parameters found are used to report baseline WER for shallow fusion.

- Tune α and β, minimizing $\sum \text{WED}(y_o, y_{\text{gt}})$. These parameters are used in non-diverse rescoring experiments.

- Tune α, β, λ and $|G|$, minimizing $\sum \text{WED}(y_o, y_{\text{gt}})$. We use these parameters in the diverse beam search rescoring experiments.

The total WED is measured for each batch, and combinations performing worse than the median are pruned. Including pruned attempts, we perform 40 trials for each combination, noting few or no improvements beyond the first 20 trials. All searches are run on an evaluation set.

Using the optimal beam search parameters, we score all candidates in Y using BERT. We then perform a grid search for $\gamma \in [0, 0.5]$ with step size 0.001 to find the interpolation parameter minimizing the total WED on the evaluation set.

5.7 Implementation details

As already described, parts of our system had viable components already available. Most of these were written in Python or had Python bindings available, so Python was a natural choice as the implementation language. Unfortunately, multithreading in Python is severely limited due to the global interpreter lock. The GIL forces us to either use data-parallel processing or C++/Rust modules to achieve performant parallelization.

In the critical path of the system's inference functionality, we took care to use C++ and Pytorch whenever possible. Figure 5.3 shows an icicle diagram showing time spent in various parts of the pipeline. As we did not maintain a Python implementation for comparison, we can only measure the performance improvements anecdotally in terms of the places we did not have time to optimize.

The only remaining step where significant inference processing happens in Python code is the transition between the C++ beam search module and BERT's tokenizer. In this step, the beam search paths (encoded as vocabulary indices) are decoded into Python strings for use with the tokenizer. In the final version of the codebase, this step takes longer than the actual beam search. This code is still serial, as the overhead of distributing the work across different Python processes outweighs the runtime of the actual computation.

In the training modules, we primarily relied on data-parallel multiprocessing in Python. As we ran on Telenor Research's HPC infrastructure, we achieved significant speedups by distributing the work in large chunks across many processor cores. By caching intermediate results to disk in full, we could easily and quickly run experiments with downstream components without running the entire pipeline.

5.8 Infrastructure

All computations were run on a shared HPC system hosted by Telenor Research. As this was a shared system without strict time-sharing policies, the runtimes of our different model variations are not comparable. Nevertheless, we did observe that increasing the number of groups G or n-grams consistently slowed down the beam search slightly, even though we cannot quantify this accurately.

We allocated three Nvidia RTX 2080 Ti cards for training BERT. The performance benefits of allocating more GPUs were limited. With more GPUs, the

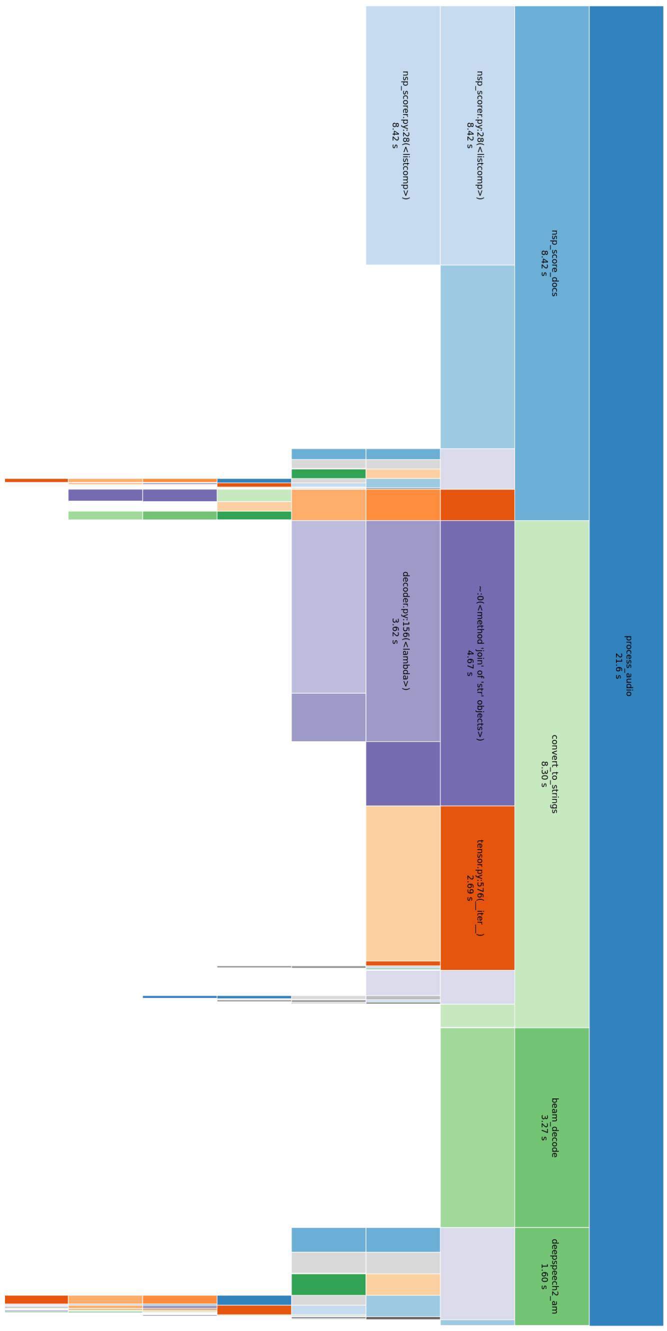


Figure 5.3: Icicle diagram showing time spent in each component when decoding 20 random samples. While it is clear that BERT (`nsp_score_docs`) slows down the system significantly, converting the output of the beam search to Python-compatible strings takes almost as long time.

model needs to process more data to converge due to increased batch size or incur additional parameter synchronization overhead with smaller batches. During inference, we ran BERT on 1-3 of the cards, depending on availability. Similar to what we observed when training, data copying overhead limited the performance benefits of allocating more than 3 GPUs.

Chapter 6

Results and Discussion

After training on the NST and TNCS datasets, the baseline system with a 2-gram LM integrated using shallow fusion obtains 33.27% WER on the TNCS test set. Rescoring the 1024 beams of that model with our best BERT model obtains 32.79% WER. This is a relative improvement of 1.4%, or 5.7% WER recovery. Repeating the experiment on NPSC data, the best BERT model obtains 20.8% WER, corresponding to a WER recovery of 37.1%. Increasing the context size for NPSC data reduces the WER further to 20.6%.

6.1 N-best rescoring

Table 6.1 shows rescoring performance with the various BERT rescoring - and training strategies. The conversational NSP training task does not help performance at all (see why in section 6.5), despite being the only training scheme where we could fine-tune on the large OpenSubtitles dataset.

On the other hand, the disambiguation task improves performance significantly. This shows that more realistic training of attention LMs does matter. In line with Devlin et al. 2019, we found that BERT does learn a sufficient language representation to quickly adapt to new writing styles and language use with only minimal fine-tuning. The disambiguation models in table 6.1 are all pre-trained on the Colossal Norwegian Corpus, and then fine-tuned on just the two conversational speech corpora[1]. We found that fine-tuning on OpenSubtitles CNSP does not improve performance and, in some cases, even hurts performance.

[1]When testing on NPSC, we fine-tune BERT on TNCS first, and the other way around for TNCS, see section 5.4.2 for details.

	TNCS		NPSC	
Decoding strategy	**WER**	**WERR**	**WER**	**WERR**
Greedy decode	48.9	-	37.7	-
BS w/ vocabulary	38.4	-	30.7	-
BS w/ 2-gram	33.3	-	23.4	-
+ Conversational NSP	33.3	0.0	-	-
+ Disambiguation no ctx	33.1	2.3	22.5	12.0
+ Disambiguation short ctx	**32.8**	5.7	22.5	12.0
+ Disambiguation long ctx	33.1	1.5	**20.8**	37.1
+ *Oracle rescorer*	*24.9*	*100.0*	*16.3*	*100.0*
BS w/ 6-gram	33.0	-	23.1	-
+ Disambiguation short ctx	**32.7**	3.1	22.3	11.4
+ Disambiguation long ctx	-	-	**20.6**	34.7
+ *Oracle rescorer*	*25.8*	*100.0*	*16.1*	*100.0*

Table 6.1: Word Error Rates (WER) and WER recovery rates (WERR) obtained with the different decoding strategies. We use the total WER evaluated on the test split of each dataset, and report numbers as percentages. WERR is calculated for each block using the plain BS + n-gram model as baseline and the corresponding oracle rescorer as the gold standard.

6.1.1 Conversational context

Much of the rescoring improvements relies on the presence of conversational context. As shown in table 6.1, introducing just two utterances of conversational context improved results significantly.

For the TNCS data, extending the context to 5 utterances reduced performance both in terms of latency and error rates, as hypothesized in section 4.4.5. As we will see in section 6.5, text-only disambiguation performance also degraded on TNCS data, indicating that too long context will mislead BERT to pick an incorrect transcript.

In contrast, NPSC benefited greatly from the longer context, obtaining 37.1% and 34.7% WERR for 2- and 6-gram shallow fusion, respectively. With the longer context, the optimizer was able to choose significantly higher values for the interpolation weight γ, as shown in figure 6.1. The improvements plateaued around $\gamma \approx 0.2$, but unlike the other experiments, a much higher γ value was needed before we observed performance degradation. We attribute this to the long-form speeches being much more similar to the pre-training data. Further, the debates focus on a specific topic, making it more likely that previous context is relevant when decoding the current utterance.

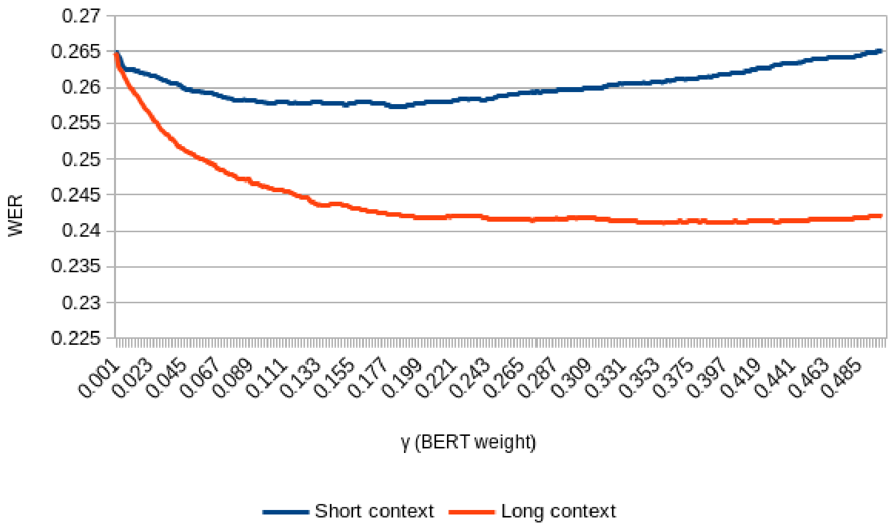


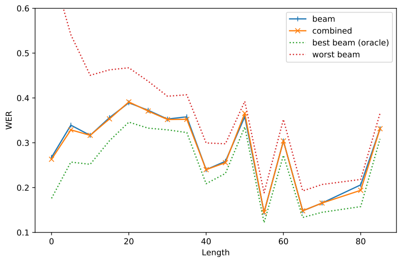
Figure 6.1: WER on the NPSC evaluation set with 2 and 5 context utterances available to BERT when rescoring. It is clear that longer context improves results on this dataset.

6.1.2 Sequence length and bounds for improvements

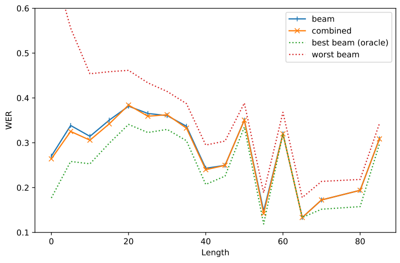
Section 4.3 discussed the upper and lower performance bounds for N-best rescoring, noting that CTC beam search will perform a near exhaustive search for short utterances. By measuring the best and worst among the candidates, we obtain an empirical estimate of these bounds.

Figure 6.2 shows this as a function of utterance length, along with the WER of y_1 and the rescored y'. The dotted lines show the total WER of the best and worst transcripts. The range between the best and worst is huge for short sequences because the beam width is large relative to the length of the sequence. This translates to a low signal-to-noise ratio because the search fills Y with garbage transcripts, hence the large values for worst WER. Still, both the n-gram and BERT are capable of picking transcripts with low WER.

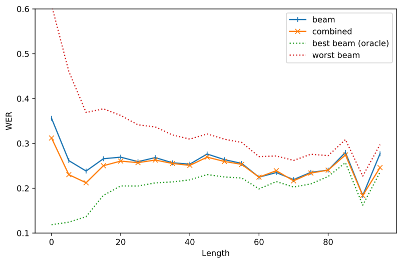
Intuitively, BERT would perform better than a 2-gram model on longer sequences because it can attend over the entire utterance, while the 2-gram is limited to two words. Figure 6.2a clearly shows there is no such relationship. If anything, BERT performs better on short utterances in figure 6.2c. When increasing the context size available to BERT from 2 to 5 utterances, figure 6.2b and 6.2d clearly show that BERT improves the results more for shorter than longer utterances. This has a simple explanation: longer context is available,



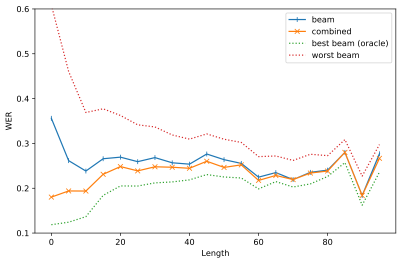
(a) TNCS short context



(b) TNCS long context



(c) NPSC short context



(d) NPSC long context

Figure 6.2: Total WER as function of utterance word count (grouped by ground truth length at 5-word intervals). "Beam" is the 2-gram baseline system. "Combined" adds a BERT model fine-tuned with 2 or 5 context utterances.

and the topic changes mentioned in the previous subsection rarely occur in short utterances.

For long sequences, however, the gap between the best and worst transcripts is tiny. This severely limits BERT's ability to improve the results and motivates our adaptation of Diverse Beam Search.

6.2 Diversity

We re-run experiments on TNCS, this time with the diverse beam search extension presented in section 4.2.1. The Optuna search found $G = 2$ and $\lambda = 10^{-6}$ to perform well (Appendix C), so the diversity bonus has a relatively small influence on the results. As described in section 3.3, diverse beam search forces the inclusion of lower-probability transcripts. Therefore, we expected the top-1 per-

LM	**Baseline**	**Diverse**
2-gram	33.27	33.29
+ BERT	32.79	33.19
+ Oracle	24.87	25.50

Table 6.2: Results on TNCS test set with standard and diverse beam search

formance to degrade when not rescoring the candidate list, as seen in table 6.2. However, the diverse variant consistently under-performed compared to standard beam search across all experiments.

A diverse beam search never performs worse than a standard beam search of width $\frac{N}{G}$ because the first group is equivalent to a beam search of that size. We find that the later groups with added diversity do not improve results. In other words, it is strictly better to spend the compute budget performing a deeper exploration of one part of the search space than forcing the search to cover larger regions.

Inspecting the beams in the second group reveals that they were often filled with seemingly random characters. The most likely explanation is that the distance metric is too simple, and the model exploits this to return garbage transcripts obtaining high diversity scores. We found the search to be extremely sensitive to the exact value of λ, with tiny changes tipping the later groups from containing homogeneous beams as in standard beam search to mainly containing garbage.

6.3 Shallow fusion vs. rescoring

To see the effect of rescoring versus shallow fusion, we ran additional experiments on the TNCS evaluation set. First, we used the 2-gram for rescoring rather than integrating it with shallow fusion. Unsurprisingly, we found that shallow fusion gives much better WER than rescoring (at the cost of being much more computationally expensive).

Then we rescored with BERT, still without shallow fusion. This yielded a WERR of 7.0%, only slightly higher than when rescoring the results from n-gram shallow fusion. While this translates to a much worse WER overall, it shows that BERT's knowledge is to a large extent distinct from the n-gram's.

Analyzing the results, BERT seems more prone to include partial words. These often occur at the end of the string, which is not surprising given that BERT's pre-training samples can contain incomplete words to fill the input window. BERT is also more tolerant when it comes to misspellings, likely because it is not constrained by a dictionary like n-gram models are. Unfortunately, this also indicates that BERT alone will not outperform an n-gram model in shallow fusion.

6.4 Qualitative results

This section examines some illustrative examples of how BERT, vocabularies, and n-gram models affect the results. Because sharing of the TNCS data is highly restricted, we mostly show examples from NPSC with similar characteristics as TNCS results.

As mentioned in section 5.5, the full-word vocabulary constraint rejects OOV words from consideration. While filtering out significant amounts of garbage, failure is impossible to recover from downstream when the correct word is OOV. With TNCS data, this frequently occurs with names and addresses. On NPSC, it happens during discussions of technical topics:

y_1	...aksept sitte gratis sa læres og på tanken i forstyrre de vassdrage
y_{gt}	...lakseparasitten gyrodactylus salaris og på kalking i forsurede vassdrag

With TNCS, some speakers tend to mumble, speak fast, use a distinct dialect, or a combination of all three. Naturally, the AM's confidence is low on those transcripts, causing the LMs to have more influence over the final transcript. To some extent, this is necessary because even a 2-gram model can effectively filter out most of the garbage words. On the other hand, the LMs prefer to fill the transcript with common words rather than what was actually said. Notice how even the oracle transcript y_o below has lost practically all semantic meaning:

c_{-1}	men det det sim kortet der skal hvert fall brukes i den ruteren der da så får du det andre sim kortet nå snart så
y_{am}	ja j ikke m notert er jeg jeg liker e ja og de skal i o et e kan fer må øre vet du jeg
y_1	ja ja ikke det er jeg jeg er da og det skal vi det er ganske mere vet du jeg
y'	ja ja ikke det er jeg jeg sier ja og det skal vi det er ganske mer vet du det
y_o	ja jeg ikke det er jeg jeg er da og det skal vi det er ganske mere vet du det
y_{gt}	ja jeg vet ikke jeg har ikke fått montert det enda jeg fikk den første i dag og det står fri montering hvem som monterer det vet du det

The above example also illustrates how conversational context is sometimes rendered useless due to topic changes (section 6.1.1). In contrast, the following example shows how BERT occasionally uses context to infer that the correct transcript would be the one using “studieplass” in plural form:

c_{-2}	og i likhet med en del andre områder hvor det er større behov for kompetanse i arbeidsmarkedet fagskoler flere som har tatt fagskoler
c_{-1}	der har det ikke kommet flere studieplasser
y_1	det har så det som vidt det er kommet noen *studieplass* på ikt
y'	det har så det så vidt det er kommet noen *studieplasser* på ikt
y_{gt}	det har så det er så vidt det er kommet noen studieplasser på i ikt

In practice, clear-cut examples of context use are surprisingly rare. If the improvements rarely come from using the context directly, why does performance improve on NPSC when we extend the conversational context? We hypothesize that context improves performance due to conditioning as discussed in section 3.1.3. Similarly, it appears that filling BERT's input window with text of similar style to the target transcript helps trigger the relevant parts of the network. If this is the case, it could be beneficial to condition on each speaker separately rather than including utterances by both speakers as context in the input window.

The grammatical changes BERT makes are often improvements over the n-gram and beam search, as shown in the next example. While they most often bring the transcript closer to the ground truth, they also introduce further deviations and sometimes distort the original meaning:

y_1	og ikke minst er det et viktig poeng dette at det *klarte* mange som som her ikke alltid veit at det er nok ulovlig som er viktig *har med segre*
y'	og ikke minst er det et viktig poeng dette at det *klart er* mange som som her ikke alltid veit at det er nok ulovlig som er viktig *å ha med seg*
y_{gt}	og ikkje minst **så er det eit** viktig poeng dette at det **er klart det er** mange som som her ikkje alltid veit **at dei gjer noko** ulovleg som er viktig å ha med seg

6.4.1 Model bias

The last example of the previous section highlights the problem of model bias. From a modeling perspective, the entire purpose of introducing LM constraints into an ASR system is to bias the results in favor of coherent, meaningful, and orthographically correct language. This reduces the model's variance in the sense that it is prevented from outputting certain words or phrases.

Usually, those phrases or words are meaningless garbage. Sometimes, however, the biasing changes the utterance's meaning. Notice in the previous section how the meaning changed from "dei gjer noko ulovleg" ("they are doing something illegal") to the statistically far more common "det er nok ulovlig" ("it probably is illegal"). In extreme cases, the system completely ignores the actual audio and replaces e.g. "nydelig" with "nei" or "ok" even though they are entirely implausible from a phonetic perspective.

Rephrased, LMs hide the AM's weaknesses. While this is good in many cases because it guesses correctly, the resulting failure mode of plausible-looking sentences is much harder for humans to detect. Whether this is a problem in practice depends on the application. However, it is crucial to be aware of these limitations when considering whether to apply strong LM constraints to an ASR system.

6.5 Analysis of BERT training strategies

We have explored several different training schemes for BERT. Our findings are summarized in table 6.4.

The NSP objective is unsuited to use for rescoring. The base model predicts that virtually no candidates are likely next sentences. In a sense, this is correct in terms of the NSP task because deviations in language use would be a telltale sign of a negative sample. Conversational NSP does not improve the situation, and is not even reliably able to avoid the same failure mode. While it shows decent performance on the training set, it is clear from table 6.4 that this is due to overfitting.

Training data/scheme	**Accuracy (%)**	**PPV (%)**	**NPV (%)**
Base model NSP	47.95	92.27	3.63
TNCS CNSP	52.33	52.98	51.68
OpenSubtitles (cased) CNSP	50.56	3.35	97.77
OpenSubtitles CNSP	48.98	15.46	82.50
+ NPSC Disambiguation y_o	73.93	80.54	67.32
+ TNCS Disambiguation y_o	81.94	88.83	75.05
TNCS Disambiguation y_o	80.96	91.71	70.20
NPSC Disambiguation y_o	74.86	87.24	62.48
+ TNCS Disambiguation y_o	**82.17**	89.94	74.39
+ TNCS Disambiguation y_{gt}	79.05	81.25	75.00

Table 6.4: Text-only disambiguation results on a balanced TNCS evaluation set with 2-utterance context, comparable to setting $N = 2$ and $\gamma = 1$. All models start from the same base model (see section 5.1.4), but are trained on different datasets until the early stopping criterion is met. PPV/NPV is the positive/negative predictive values, i.e. portion of samples predicted as positive/negative that actually are true positives/negatives.

LM	**Acc. (%)**
Base	53.13
OpenSubtitles	51.56
NPSC	59.38
TNCS	68.75
Human	70.31

Table 6.5: Accuracy on 64 evaluation samples from a balanced TNCS conversational NSP set. The human was not fine-tuned on this task.

Training on the disambiguation task, even though little data is available, leads to much better performance. While performance is better when the training data is from the same dataset as the evaluation set, we observe a significant uplift in performance from training only on NPSC. As expected, training on more conversational data consistently improves results.

6.5.1 Conversational NSP and human performance

Our conversational NSP models performed rather poorly both on the disambiguation task and for rescoring. As shown in table 6.5, the base model did not perform much better than a random guess, despite being trained on a relatively similar task. The accuracy did not improve until BERT was exposed to a transcription dataset.

When we attempted the conversational NSP task ourselves, we found it to be surprisingly tricky. While some samples were easy to classify, most required heavy use of domain knowledge, both in the form of common patterns in phone conversations and facts specific to telecommunications (i.e. “no internet” likely precedes a question about the router).

We assume this is the reason behind the considerable performance improvement when training on TNCS data. Therefore, we conclude that conversational NSP is unlikely to improve further, especially for TNCS where we found little potential for improvement by extending the context (section 6.1.1).

Chapter 7

Conclusion

As part of an ongoing project to improve ASR for real-life Norwegian conversations, we have explored how state-of-the-art language models can improve performance. Working towards that goal, we have successfully trained BERT to directly disambiguate the outputs of a CTC beam search. In terms of the research questions we posed in section 1.2:

RQ1: How do state-of-the-art LM integration methods perform in low-resource situations?

We implemented N-best rescoring with BERT into Telenor's ASR pipeline. Even though this method is conceptually very simple, it yielded significant performance improvements. While the results were comparable to those reported by the broader research community, the analysis revealed significant potential for further performance gains by improving the decoding process. This indicates that deeper integration of large LMs is a promising direction for future research.

RQ2: How can LMs exploit conversational context to improve performance?

Despite having relatively few conversational speech transcripts available for training, ASR performance improved significantly when introducing conversational context. Our results suggest that BERT makes heavy use of previous utterances to achieve performance gains over the 2-gram LM for the formal NPSC dataset.

RQ3: How does the LM training scheme and -data affect results?

The experiments confirmed that the choice of LM training task had a significant effect on the results. BERT learned to perform all proposed training tasks in isolation, and to some extent, even successfully generalize across datasets. However, both traditional NSP and conversational NSP performed poorly when the model was integrated as part of an ASR pipeline. Training BERT to directly disambiguate between candidate transcripts gave the best performance.

When it comes to datasets, the deciding factor was the modality of the dataset. BERT performed much better when trained on speech transcripts than when trained on movie subtitles and text data. Additional CNSP pre-training on movie subtitles even degraded performance in several cases.

7.1 Future work

Given the modest performance improvements observed on TNCS data with N-best rescoring, there is room for future improvements. We recommend three lines of research as natural next steps beyond the obvious solution of improving the AM itself.

7.1.1 Tighter LM integration

The most obvious area to improve would be to find a way to use BERT with shallow fusion without requiring large amounts of fine-tuning data. We have already seen that n-grams perform much better when given a chance to guide the beam search, and there is no apparent reason why this should not be the case with BERT too. By (for example) introducing a token indicating incomplete sentences, BERT should be able to learn to operate correctly even though the utterance is not fully decoded. An autoregressive, generative transformer model could also be an option, though recent work in this area have to a large extent focused on high-resource situations.

7.1.2 Context

In regards to conversational context, we have shown that language models can make use of the conversational context when rescoring. It would be interesting to explore how context can be used more efficiently, especially in the setting of informal conversations. We forced BERT to treat the conversations as linear, even when the speech was in practice overlapping. As CTC often outputs each token approximately at the time step it was uttered, it is possible to align each word in a conversation and use BERT's segment embedding to indicate the speaker. This gives BERT a higher-fidelity context to work with.

The experiments did not show exactly which aspects of the conversational context BERT exploits. Future research could explore this theme further to see if speaker context can substitute conversational context. If BERT is shown to use the context for conditioning as briefly hypothesized in section 6.4, other in-domain context could be used, leading to improved performance beyond conversational ASR.

7.1.3 Vocabulary

A limiting factor we encountered was the mismatch between the character-level acoustic model and beam search, word-level n-gram model, and subword-level neural LM. Ensuring all components have a shared vocabulary would likely improve runtime performance and make tighter LM integration far more straightforward. This vocabulary would need to be of limited size, as a too large vocabulary makes it difficult for the AM to tie all words with their pronunciations correctly. At the same time, BERT benefits from larger vocabularies, so a good trade-off needs to be found.

A shared vocabulary is also a prerequisite to implementing both deep and cold fusion. On the one hand, this could improve performance because the AM does not encounter enough training data to learn a good internal LM. On the other hand, it is an open question whether the low amount of training data available makes it too difficult for the AM to learn how to use the integrated LM efficiently while simultaneously adapting the LM to spoken language.

7.1.4 Acoustic Model

Finally, we emphasize that relying heavily on language models is merely a workaround to the underlying problem of poor acoustic model performance. This workaround has some adverse side effects, particularly model bias and less explainable/detectable failure modes.

While AMs are competitive with human performance in high-resource domains and on "easy" datasets such as NST, the combination of little training data and spontaneous speech gives inferior results. Therefore, an obvious direction for future research is to improve the AM itself in this situation. Rescoring is shown to improve results even with the combination of small NLMs and robust AMs, though the room for improvement becomes smaller for better-performing baselines.

Bibliography

Akiba, Takuya et al. (July 25, 2019). "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '19. New York, NY, USA: Association for Computing Machinery, pp. 2623–2631. ISBN: 978-1-4503-6201-6. DOI: 10.1145/3292500.3330701. URL: https://doi.org/10.1145/3292500.3330701 (visited on 05/22/2021).

Amodei, Dario et al. (June 11, 2016). "Deep Speech 2 : End-to-End Speech Recognition in English and Mandarin". In: *International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 1938-7228. PMLR, pp. 173–182. URL: http://proceedings.mlr.press/v48/amodei16.html (visited on 06/14/2021).

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (May 19, 2016). "Neural Machine Translation by Jointly Learning to Align and Translate". In: *arXiv:1409.0473 [cs, stat]*. arXiv: 1409.0473. URL: http://arxiv.org/abs/1409.0473 (visited on 10/13/2020).

Bengio, Y., P. Simard, and P. Frasconi (Mar. 1994). "Learning long-term dependencies with gradient descent is difficult". In: *IEEE Transactions on Neural Networks* 5.2. Conference Name: IEEE Transactions on Neural Networks, pp. 157–166. ISSN: 1941-0093. DOI: 10.1109/72.279181.

Bojanowski, Piotr et al. (June 1, 2017). "Enriching Word Vectors with Subword Information". In: *Transactions of the Association for Computational Linguistics* 5, pp. 135–146. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00051. URL: https://doi.org/10.1162/tacl_a_00051 (visited on 06/14/2021).

Brown, Tom B. et al. (July 22, 2020). "Language Models are Few-Shot Learners". In: *arXiv:2005.14165 [cs]*. arXiv: 2005.14165. URL: http://arxiv.org/abs/2005.14165 (visited on 11/03/2020).

Chan, William et al. (Mar. 2016). "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition". In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2016 IEEE International Conference on Acoustics, Speech and Signal Processing

(ICASSP). ISSN: 2379-190X, pp. 4960–4964. DOI: 10.1109/ICASSP.2016.7472621.

Chen, Stanley F. and Joshua Goodman (Oct. 1, 1999). “An empirical study of smoothing techniques for language modeling”. In: *Computer Speech & Language* 13.4, pp. 359–394. ISSN: 0885-2308. DOI: 10.1006/csla.1999.0128. URL: http://www.sciencedirect.com/science/article/pii/S0885230899901286 (visited on 12/04/2020).

Cho, Jaejin et al. (May 2019). “Language Model Integration Based on Memory Control for Sequence to Sequence Speech Recognition”. In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). ISSN: 2379-190X, pp. 6191–6195. DOI: 10.1109/ICASSP.2019.8683380.

Clark, Kevin et al. (Mar. 23, 2020). “ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators”. In: *arXiv:2003.10555 [cs].* arXiv: 2003.10555. URL: http://arxiv.org/abs/2003.10555 (visited on 09/21/2020).

Dai, Zihang et al. (July 2019). “Transformer-XL: Attentive Language Models beyond a Fixed-Length Context”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics.* ACL 2019. Florence, Italy: Association for Computational Linguistics, pp. 2978–2988. DOI: 10.18653/v1/P19-1285. URL: https://www.aclweb.org/anthology/P19-1285 (visited on 06/14/2021).

Davis, K. H., R. Biddulph, and S. Balashek (Nov. 1, 1952). “Automatic Recognition of Spoken Digits”. In: *The Journal of the Acoustical Society of America* 24.6. Publisher: Acoustical Society of America, pp. 637–642. ISSN: 0001-4966. DOI: 10.1121/1.1906946. URL: https://asa.scitation.org/doi/abs/10.1121/1.1906946 (visited on 11/09/2020).

Dehghani, Mostafa et al. (2019). “Universal Transformers”. In: URL: https://openreview.net/pdf?id=HyzdRiR9Y7 (visited on 06/14/2021).

Devlin, Jacob et al. (May 24, 2019). “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *arXiv:1810.04805 [cs].* arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805 (visited on 10/13/2020).

Futami, Hayato et al. (Aug. 9, 2020). “Distilling the Knowledge of BERT for Sequence-to-Sequence ASR”. In: *arXiv:2008.03822 [cs, eess].* arXiv: 2008.03822. URL: http://arxiv.org/abs/2008.03822 (visited on 10/01/2020).

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning.* http://www.deeplearningbook.org. MIT Press.

Graves, Alex et al. (Jan. 1, 2006). “Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks”. In: URL: http://www.cs.toronto.edu/~graves/icml_2006.pdf.

Gulati, Anmol et al., eds. (2020). *Conformer: Convolution-augmented Transformer for Speech Recognition.*

Gulcehre, Caglar et al. (June 12, 2015). “On Using Monolingual Corpora in Neural Machine Translation”. In: *arXiv:1503.03535 [cs]*. arXiv: `1503.03535`. URL: `http://arxiv.org/abs/1503.03535` (visited on 04/14/2021).

Hannun, Awni et al. (Dec. 19, 2014). “Deep Speech: Scaling up end-to-end speech recognition”. In: *arXiv:1412.5567 [cs]*. arXiv: `1412.5567`. URL: `http://arxiv.org/abs/1412.5567` (visited on 11/03/2020).

Hannun, Awni Y. et al. (Dec. 8, 2014). “First-Pass Large Vocabulary Continuous Speech Recognition using Bi-Directional Recurrent DNNs”. In: *arXiv:1408.2873 [cs]*. arXiv: `1408.2873`. URL: `http://arxiv.org/abs/1408.2873` (visited on 10/13/2020).

Harris, Zellig (1954). “Distributional Structure”. In: *Word* 10:2.3, pp. 146–162. URL: `https://zelligharris.org/Distributional.Structure.pdf`.

Heafield, Kenneth et al. (Aug. 2013). “Scalable Modified Kneser-Ney Language Model Estimation”. In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 690–696. URL: `https://www.aclweb.org/anthology/P13-2121`.

Hochreiter, S. and J. Schmidhuber (Nov. 15, 1997). “Long short-term memory”. In: *Neural Comput* 9.8, pp. 1735–1780. ISSN: 0899-7667. DOI: `10.1162/neco.1997.9.8.1735`.

Huang, M. et al. (Dec. 2019). “Exploring Model Units and Training Strategies for End-to-End Speech Recognition”. In: *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. 2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), pp. 524–531. DOI: `10.1109/ASRU46091.2019.9003834`.

Kamath, Uday, John Liu, and James Whitaker (2019). *Deep Learning for NLP and Speech Recognition.* Cham: Springer International Publishing. ISBN: 978-3-030-14596-5. DOI: `10.1007/978-3-030-14596-5_12`. URL: `https://doi.org/10.1007/978-3-030-14596-5_12`.

Lan, Zhenzhong et al. (Feb. 8, 2020). “ALBERT: A Lite BERT for Self-supervised Learning of Language Representations”. In: *arXiv:1909.11942 [cs]*. arXiv: `1909.11942`. URL: `http://arxiv.org/abs/1909.11942` (visited on 11/10/2020).

LeCun, Y. et al. (Dec. 1, 1989). “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Computation* 1.4, pp. 541–551. ISSN: 0899-7667. DOI: `10.1162/neco.1989.1.4.541`. URL: `https://doi.org/10.1162/neco.1989.1.4.541` (visited on 04/28/2021).

Lison, Pierre and Jörg Tiedemann (May 2016). “OpenSubtitles2016: Extracting Large Parallel Corpora from Movie and TV Subtitles”. In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation*

(LREC 2016). Ed. by Nicoletta Calzolari (Conference Chair) et al. Portorož, Slovenia: European Language Resources Association (ELRA). ISBN: 978-2-9517408-9-1.

Liu, Alexander H., Hung-yi Lee, and Lin-shan Lee (May 2019). "Adversarial Training of End-to-end Speech Recognition Using a Criticizing Language Model". In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). ISSN: 2379-190X, pp. 6176–6180. DOI: `10.1109/ICASSP.2019.8683602`.

Liu, Yinhan et al. (July 26, 2019). "RoBERTa: A Robustly Optimized BERT Pretraining Approach". In: *arXiv:1907.11692 [cs]*. arXiv: `1907.11692`. URL: `http://arxiv.org/abs/1907.11692` (visited on 09/21/2020).

Ma, Jeff and Richard Schwartz (Jan. 1, 2008). "Unsupervised versus supervised training of acoustic models." In: pp. 2374–2377.

Olah, Christopher (Aug. 27, 2015). *Understanding LSTM Networks*. colah's blog. URL: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/` (visited on 11/17/2020).

Pennington, Jeffrey, Richard Socher, and Christopher D. Manning (2014). "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543. URL: `http://www.aclweb.org/anthology/D14-1162`.

Rogers, Anna, Olga Kovaleva, and Anna Rumshisky (Feb. 27, 2020). "A Primer in BERTology: What we know about how BERT works". In: *arXiv:2002.12327 [cs]*. arXiv: `2002.12327`. URL: `http://arxiv.org/abs/2002.12327` (visited on 09/21/2020).

Røyneland, Unn et al. (Nov. 21, 2018). *Språk i Norge – kultur og infrastruktur*. URL: `https://sprakinorge.no/` (visited on 10/13/2020).

Schuster, M. and K. Nakajima (Mar. 2012). "Japanese and Korean voice search". In: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). ISSN: 2379-190X, pp. 5149–5152. DOI: `10.1109/ICASSP.2012.6289079`.

Shibata, Yusuke et al. (Sept. 10, 1999). "Byte Pair Encoding: A Text Compression Scheme That Accelerates Pattern Matching". In:

Shin, Joonbo, Yoonhyung Lee, and Kyomin Jung (Oct. 15, 2019). "Effective Sentence Scoring Method Using BERT for Speech Recognition". In: *Asian Conference on Machine Learning*. Asian Conference on Machine Learning. ISSN: 2640-3498. PMLR, pp. 1081–1093. URL: `http://proceedings.mlr.press/v101/shin19a.html` (visited on 06/14/2021).

Solli, Arne (1998). *Nordic() - funksjon for soudex-tilpassing av norske namn.* URL: `https://folk.uib.no/hhiso/avhandling/progdok/vb/nordic.htm` (visited on 11/25/2020).

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le (Dec. 14, 2014). "Sequence to Sequence Learning with Neural Networks". In: *arXiv:1409.3215 [cs]*. version: 3. arXiv: `1409.3215`. URL: `http://arxiv.org/abs/1409.3215` (visited on 11/26/2020).

Synnaeve, Gabriel et al. (July 14, 2020). "End-to-end ASR: from Supervised to Semi-Supervised Learning with Modern Architectures". In: *arXiv:1911.08460 [cs, eess]*. arXiv: `1911.08460`. URL: `http://arxiv.org/abs/1911.08460` (visited on 10/13/2020).

Toshniwal, Shubham et al. (Dec. 2018). "A Comparison of Techniques for Language Model Integration in Encoder-Decoder Speech Recognition". In: *2018 IEEE Spoken Language Technology Workshop (SLT)*. 2018 IEEE Spoken Language Technology Workshop (SLT), pp. 369–375. DOI: `10.1109/SLT.2018.8639038`.

Vaswani, Ashish et al. (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc. URL: `https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

Vijayakumar, Ashwin et al. (2018). "Diverse beam search for improved description of complex scenes". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. URL: `https://aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17329`.

Wolf, Thomas et al. (2019). "HuggingFace's Transformers: State-of-the-art Natural Language Processing". In: *ArXiv* abs/1910.03771.

Xiong, W. et al. (Aug. 24, 2017). "The Microsoft 2017 Conversational Speech Recognition System". In: *arXiv:1708.06073 [cs]*. arXiv: `1708.06073`. URL: `http://arxiv.org/abs/1708.06073` (visited on 12/02/2020).

Yun, Chulhee et al. (Feb. 24, 2020). "Are Transformers universal approximators of sequence-to-sequence functions?" In: *arXiv:1912.10077 [cs, stat]*. arXiv: `1912.10077`. URL: `http://arxiv.org/abs/1912.10077` (visited on 10/13/2020).

Appendix A

Conversations from OpenSubtitles

In this thesis, we discuss distinctions between written and spoken language. The OpenSubtitles dataset described in section 5.1.5 exists in a middle ground: it uses spoken language in the sense that it is intended to be read out loud, but since it is intended to be shown onscreen and read quickly, the speech is compressed and details are omitted for readability. Such edits typically lead to text that reads more like written language.

Though the OpenSubtitles dataset was barely mentioned in chapter 6, it was still used in several experiments. For completeness, this appendix outlines some of our findings with the OpenSubtitles data.

A.1 Pre-training on conversational text

Using a RoBERTa (Y. Liu et al. 2019) model, we pre-train on the Masked Language Modeling (MLM) task with both OSCAR[1] and OpenSubtitles data, and measure the loss on both datasets as well as the TNCS transcripts. Results are shown in figure A.1.

After pre-training on OSCAR, we observe that the loss is much higher on OpenSubtitles than on OSCAR. As we pre-train further on OpenSubtitles, the loss on OSCAR increases while the loss on OpenSubtitles decreases. This shows that the language use in the two datasets are quite distinct, and that RoBERTa struggles to find a single set of language representations appropriate for both datasets.

[1]`https://web.archive.org/web/20200928211119/https://oscar-corpus.com/`

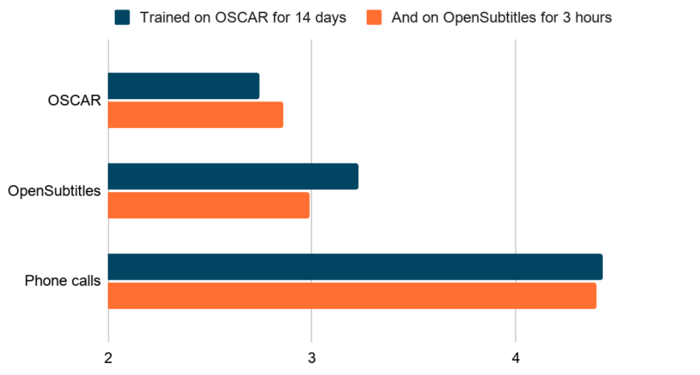


Figure A.1: MLM loss on each dataset before and after additional pre-training on OpenSubtitles.

Pre-trained on	**WER (%)**	**Accuracy (%)**
OSCAR	1.35	89.46
+OpenSubtitles	1.37	89.32

Table A.1: Text-only disambiguation WER and accuracy on TNCS before and after pre-training on OpenSubtitles conversations. Results were run with 4 utterances of bidirectional context and 2 candidates to disambiguate, one of which were the ground truth transcript.

Observing the loss on TNCS transcripts, we see that the loss decreased when training on OpenSubtitles. Intuitively, this would indicate that the language representations learned from OpenSubtitles is more suitable for phone transcripts than those learned from OSCAR. On the other hand, we observed that the disambiguation performance on TNCS degrades as shown in table A.1, contradicting the previous observation.

A.2 Audio-free disambiguation

Since OpenSubtitles contains conversations, it should be relevant for pre-training on the disambiguation task. During development of the disambiguation pipeline, we also used OpenSubtitles for testing as it is easier to debug locally.

The disambiguation task can be adapted to work with text-only datasets. We define a function Randomize(y, p, k) which outputs k variations of y with a fraction $p \in (0, 1)$ of the graphemes replaced with phonetically similar alternatives.

	2 candidates		51 candidates	
	WER (%)	Accuracy (%)	WER (%)	Accuracy (%)
RoBERTa	0.03	98.68	0.34	83.03
6-gram	0.04	97.24	0.58	71.02

Table A.2: Results on OpenSubtitles disambiguation with 2 lines of context.

Adjusting the optimization target, the text-only disambiguation task would be to minimize

$$\min_{\theta} \sum_{(c,y_{\text{gt}})\in\mathcal{D}} \left[\mathcal{L}\left(1, P_{LM_\theta}(y_{\text{gt}}|c)\right) + \sum_{y^*\in\text{Randomize}(y_{\text{gt}},p,k)} \mathcal{L}\left(0, P_{LM_\theta}(y^*|c)\right) \right].$$

As before, $\mathcal{L}$ refers to the cross-entropy loss function.

A straightforward algorithm to identify phonetically similar words is Soundex. We adapt an implementation by Solli 1998, which also works well on general text. Since s is a many-to-one function, replacements for Randomize can be obtained easily by computing $s^{-1}(s(x))$. Setting $p = 10\%$ and $k = 1$ and $k = 50$, we train and evaluate RoBERTa's performance on OpenSubtitles disambiguation.

Results are shown in table A.2. It is clear from the table that the task is far too easy for the model, with near-perfect performance on the balanced dataset. Qualitatively, we observe that the mistakes made by this candidate generator are easy to spot for a human evaluator. Since this task appears to be too easy, we do not pursue this research direction further at this time.

Appendix B

Scaling BERT-like models

The work in this thesis is a continuation from the specialization project TDT4501. Since the report from TDT4501 is not publicly available, we briefly summarize some relevant findings regarding scalability in this appendix. The main goal of the specialization project was to determine whether it would be worth integrating large-scale Neural Language Models (NLMs) in low-resource domains. This was done by pre-training several LM types on a range of standard pre-training tasks. All evaluations are based on a text-only version of the disambiguation task (i.e. $\gamma = 1$), unlike the results in the main text. We consider two aspects of scaling: increased difficulty and computational complexity.

Since the AM is trained to give a higher ranking to correct transcripts, introducing more transcripts to the disambiguation task causes potentially higher WER when picking the wrong one. In addition, more choices increase the number of incorrect alternatives, further increasing the potential performance loss. Despite the worst-case scenario being much worse than before, figure B.1 shows that all models retain most of their performance and that the performance degradation is more or less the same across all models.

The second aspect of scaling up disambiguation is the amount of compute required. As noted in section 4.4, RoBERTa's MLM head is expected to be much more expensive to run, and figure B.2 shows that this is indeed the case. n-gram models are both fast and yield a very competitive WER. The ELECTRA model[1] shows remarkably consistent runtime, even as the context size increases. This is enabled by GPU parallelization and the Replaced Token Detection[2]. This clearly shows the practical advantage of avoiding large output layers.

[1]Which in terms of computational complexity is similar to a NSP classification head.

[2]Binary classification of whether each token has been replaced, avoids doing T passes through the network. See details in Clark et al. 2020.

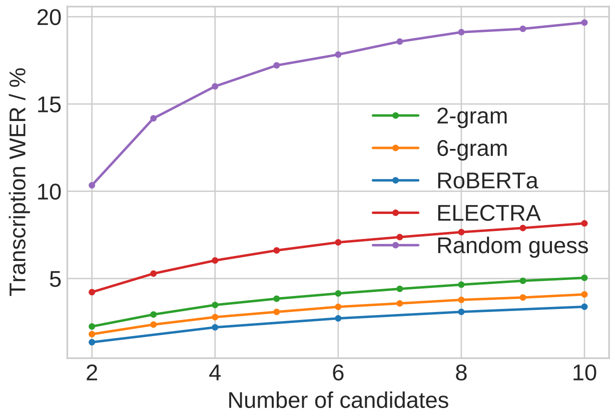


Figure B.1: WER on disambiguation task with more incorrect transcripts included. Bidirectional context size was set to 4 utterances.

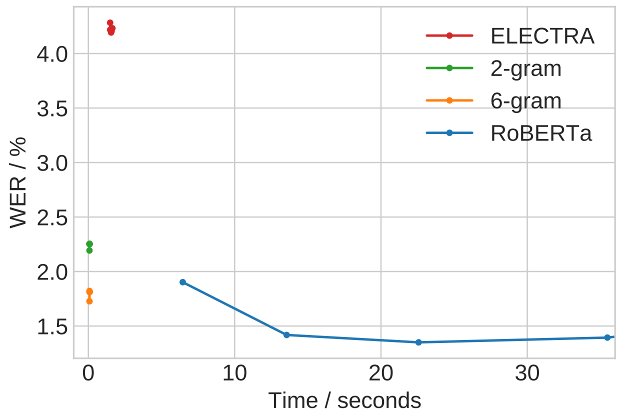


Figure B.2: WER plotted against the time required to disambiguate transcripts for each conversation. Lines follow increases in context size.

Appendix C

Hyperparameter search

In this section we show additional graphs and figures from the hyperparameter tuning.

- γ and `ext_weight` is the weight of LM2 in equation 4.4.
- `ngram_weight` is α in equation 4.1.
- `length_weight` is the word count bonus β in equation 4.1.
- `diversity_factor_exp` is $\log_2(\lambda)$ in equation 4.3.
- `num_groups_exp` is $\log_2(G)$ in equation 4.3.
- `Objective value` is $\mathrm{WER}(y_{\mathrm{gt}}, y_o)$.

In C.1, there are no values of γ that would give obvious improvements in WER. This made it hard for the hyperparameter search to set γ correctly. Comparing with C.2, it is much more obvious what value to set for γ.

Also notice how the Conversational NSP model behaved very differently between the two TNCS splits. C.1 show several failed models, including a smaller RoBERTa (Y. Liu et al. 2019) model trained on far less data, and how the garbage transcripts from the disambiguation causes BERT to output wildly incorrect results.

On the NPSC splits in figure C.3 and C.4, we see far more consistent results. It is clear that longer context consistently outperforms the other models. At the same time, the zero-context results are identical to the 2-context results. This is partially because both use the same BERT model (always trained on 2 context utterances), but it is nevertheless interesting to see the model perform equally even without the context available.

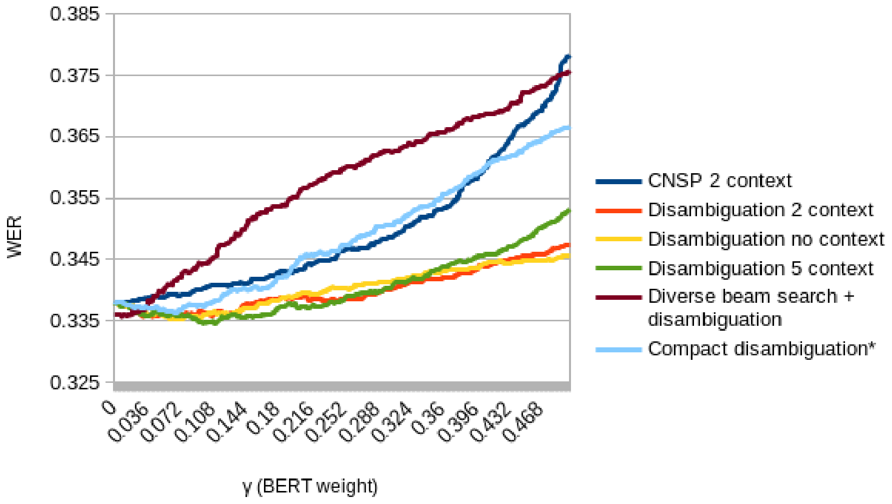


Figure C.1: WER as a function of γ on the TNCS evaluation set. (*) is a smaller RoBERTa model pre-trained on just 5GB of text.

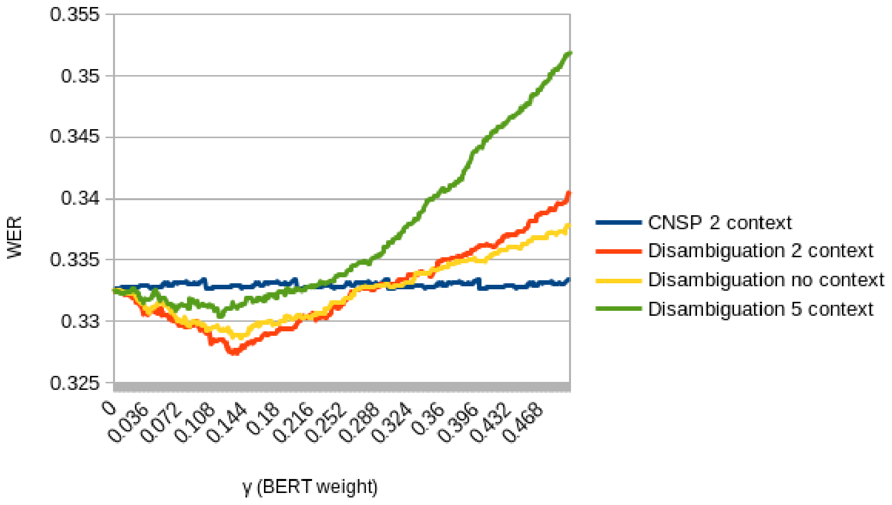


Figure C.2: WER as a function of γ on the TNCS test set.

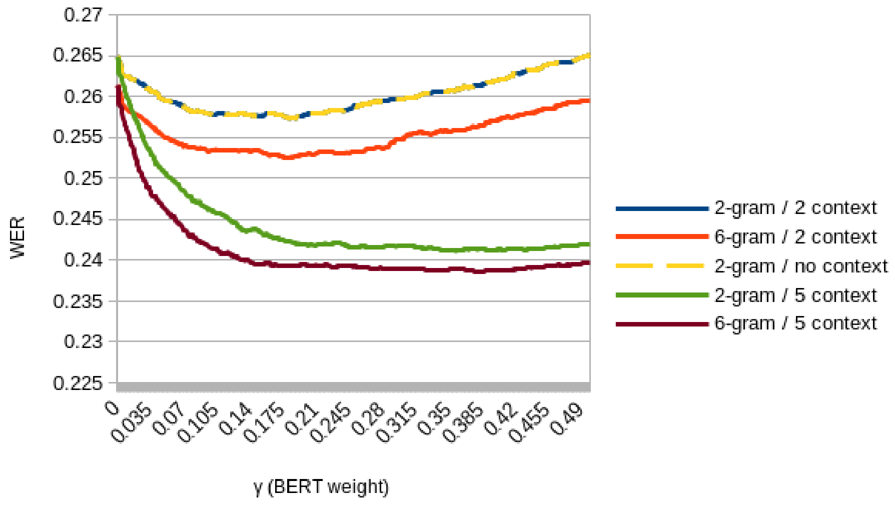


Figure C.3: WER as a function of γ on the NPSC evaluation set.

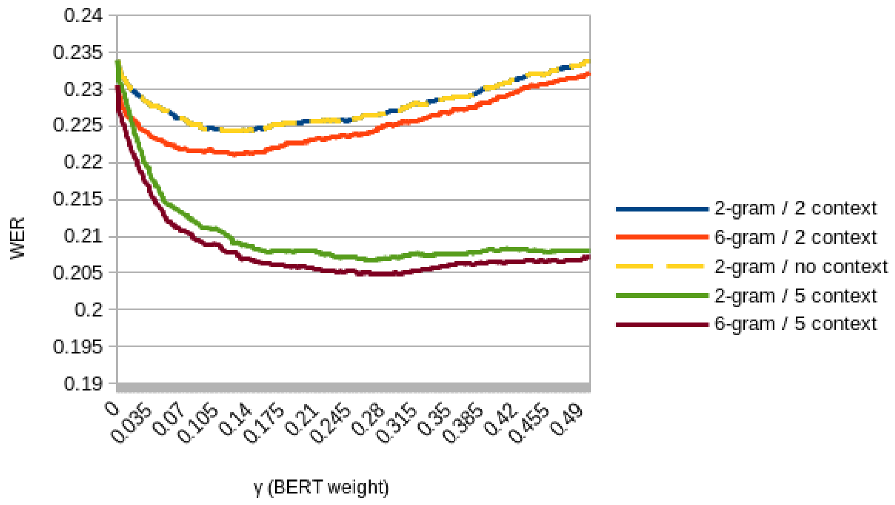


Figure C.4: WER as a function of γ on the NPSC test set.

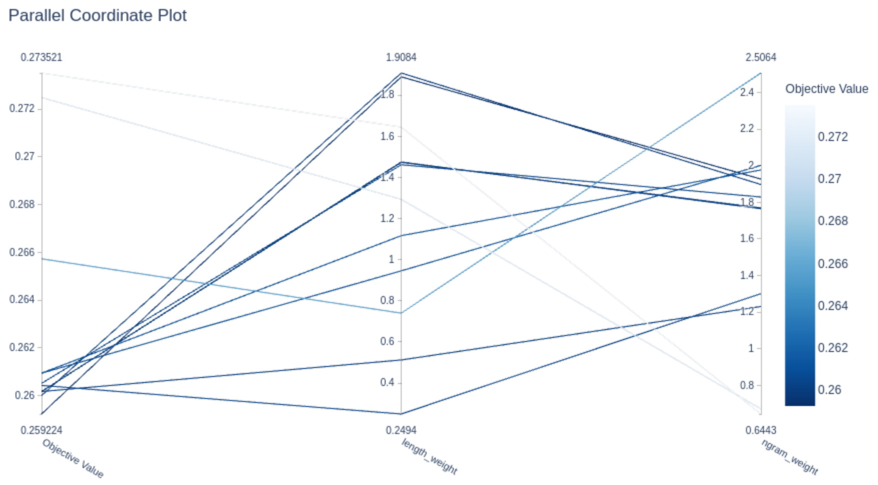


Figure C.5: Parallel coordinate plot from TNCS hyperparameter search.

The parallel coordinate plots in figure C.5, C.6 and C.8 show how hyperparameters affect the oracle WER (4.5). It is clear that the exact values are less important, especially for α and β. For the diverse beam search in figure C.6, we see clearly that the search prefers to only have two groups. This translates to throughout search of fewer parts of the search space. Nevertheless, the best objective value achieved is worse than in for the standard beam search, as reported in section 6.2. The contour plots in figure C.7 and figure C.9 tell a similar story. Note that all these plots only show completed Optuna runs – many runs were pruned and are not included in the plots.

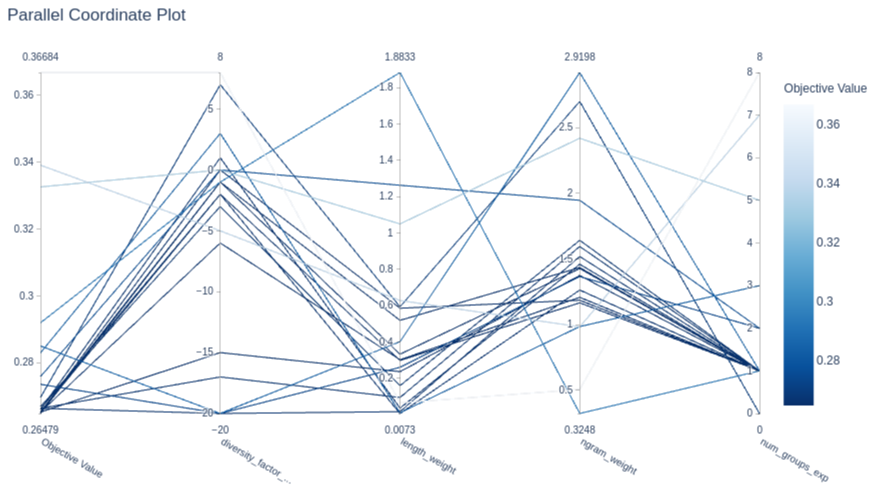


Figure C.6: Parallel coordinate plot from TNCS hyperparameter search with diverse beam search.

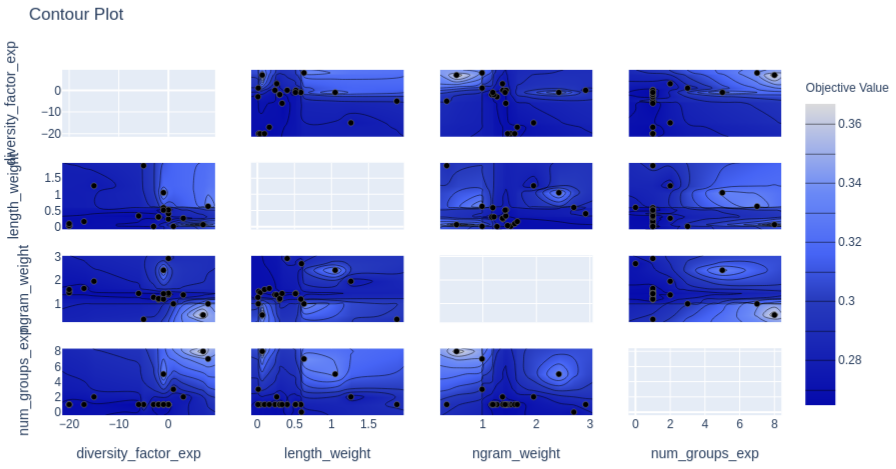


Figure C.7: Contour plot from TNCS hyperparameter search with diverse beam search.

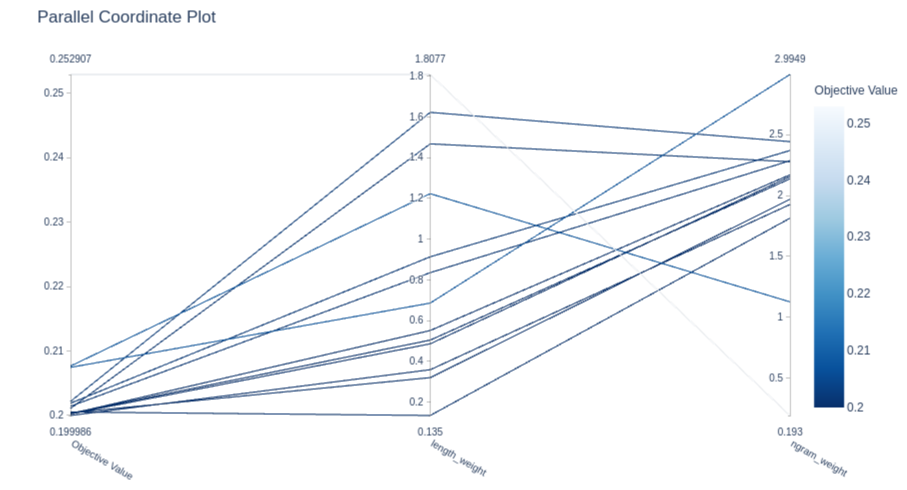


Figure C.8: Parallel coordinate plot from NPSC hyperparameter search.

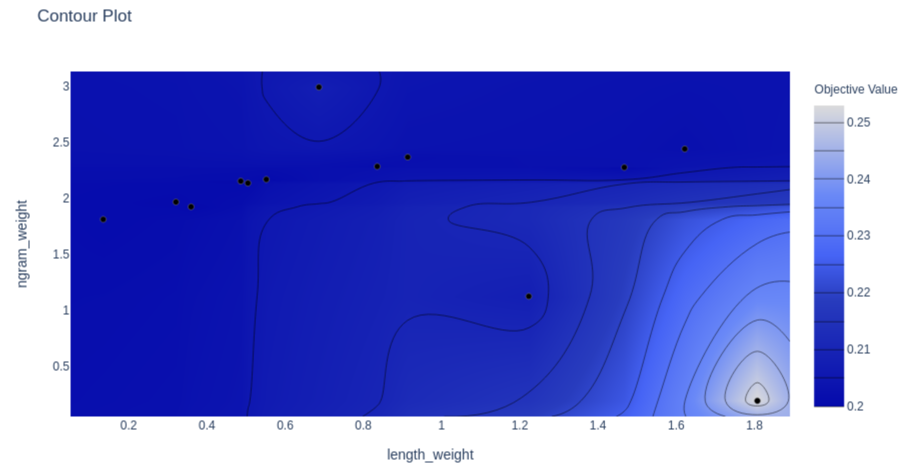


Figure C.9: Contour plot from NPSC hyperparameter search.

Simen Burud

Conversational Language Models for Low-Resource Speech Recognition

NTNU
Norwegian University of
Science and Technology

telenor