End-to-End ASR with Binarized Neural Networks

Master's Thesis | MSc. Voice Technology

Benjamin Luks
S4938712

Supervisor
Shekhar Nayak

Co-Supervisor
Matt Coler

Rijksuniversiteit Groningen | Campus Fryslân

in collaboration with

Screevo.ai

2021/2022

Acknowledgements

Thank you to the Center for Information Technology of the University of Groningen for their support and for providing access to the Peregrine high performance computing cluster. Thank you to the University of Groningen and the faculty of Campus Fryslân. Thank you to the Voice Technology team, and to my advisors, Shekhar and Matt. Your guidance has been an immeasurable help.

Thank you to my colleagues in the STT team at Screevo, Domenico, Nicola, and Zakria, for their guidance, their patronage, and for challenging the limitations of my own capabilities.

Lastly, thank you to my family for their continued support, feigning interest in a topic so obscure and foreign, and to my partner, Amarens. For better or worse, your patience and support has been unwaivering throughout this process. For that I am eternally grateful.

Contents

ABSTRACT: Binarized Neural Networks have demonstrated tremendous abilities in compressing and speeding up neural networks, with, in some cases, comparatively little degradation in performance. Despite their successful application in convolutional and feed-forward neural network units, little research has been conducted on the binarization of recurrent units. Furthermore, existing binarized recurrent neural networks have yet to be applied to end-to-end automatic speech recognition (ASR). This work, to my knowledge, marks the first attempt to apply binarized LSTM units, per (Ardakani et al., 2018), to end-to-end ASR. Experiments are conducted on networks with Connectionist Temporal Classification (CTC), as well as Attention-based architectures. Although no experiments produce high-performant, deployment-ready BNNs, greater insight into the applicability of such networks to ASR is achieved. These insights include the improved performance of linear input layers in binarized networks, as well as the importance of bidirectionality in binarized LSTMs.

Keywords: Binary Neural Networks, Neural Network Compression, Quantization, Binarized LSTM, Automatic Speech Recognition, Connectionist Temporal Classification

1 Introduction

Automatic Speech Recognition (ASR) and applications built thereupon are rapidly becoming a central part of improving the efficiency of many personal and professional tasks. From personal voice assistants, to video captioning, to air traffic control troubleshooting, its applicability is immensely diverse and ever-growing.

A limitation to the mass adoption of ASR is its hardware demand. Modern ASR technologies are often built using large, multi-million– or even billion-parameter neural networks occupying numerous gigabytes of storage and requiring graphical processing units (GPUs) in order to operate efficiently. This precludes the possibility of their being deployed directly on the devices on which they are intended to run. Instead, said devices interface with a remote server which receives the speech input and responds with the result– a text transcription in the case of speech-to-text (STT).

A number of problems exist with such systems: For one, in order to operate frictionlessly, the user must have a stable internet connection. Similarly, such frictionlessness requires that the server housing the ASR engine be within a reasonable proximity to the user. Running one, let alone several large networks on GPUs is both prohibitively expensive for smaller entities and degradative to the environment (Biewald, 2019). Furthermore, passing sensitive data across networks subjects it to security vulnerabilities.

A recent rise of interest in binarized neural networks (BNNs) has come to suggest that the abundance of required computational storage and power that these power-hungry ASR technologies demand may not be entirely necessary. BNNs are neural networks whose weights and often activations and biases are either 1 or -1. Such networks can operate using a fraction of the computational resources of traditional neural networks. Their recent success in computer vision, and, to a lesser extent, language modelling tasks suggest their viability as a technique to reduce the computational demands of a variety of neural network-based technologies.

Despite these successes, little research has been conducted into the application of BNNs to ASR. Worse yet, nearly none of this research is concerned with modern, end-to-end ASR approaches. This work aims to change that. This is done with the combination of techniques used to binarize sequential architectures as well as techniques applied to binarized hybrid ASR.

More precisely, benchmark performance has been achieved on binarized recurrent neural networks (RNNs) of the long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) variant trained on natural language processing (NLP) tasks, with a technique the authors dub *Learning Recurrent Binary/Ternary Weights*, henceforth *LRBW* (Ardakani et al., 2018). Given the precedence of successfully applying NLP architectures to ASR tasks (Chorowski et al., 2014; Dong et al., 2018), it stands to reason that this technique can be applied to end-to-end ASR networks built upon RNNs.

Although the experiments do not ultimately yield a performant BNN for end-to-end ASR, this research offers a number of contributions to the literature:

- BNNs are applied to end-to-end, LSTM-based ASR for the first time.
- The temporal dependencies which govern the preservation of information is better understood, through the comparison of single– and bidirectional LSTMs.
- The effectiveness of *LRBW* is tested on longer– and dimensionally larger inputs. The input lengths in the original LRBW experiments do not quite match those of typical speech inputs. Furthermore, the numerical density is much greater for speech features, compared with one-hot encoded text tokens. The robustness of LRBW to such factors is therefore evaluated.
- A PyTorch (Paszke et al., 2019) module is built, interchangeable with the native PyTorch LSTM cell, which can be used in training LRBW-based BNNs.[1]

Section 2 offers background into some of the mathematical and hardware technicalities of BNNs. Section 3 is a thorough literature review, summarizing recent work in ASR, and BNNs. Section 4 is an in-depth explanation of the LRBW approach to binarization. The research question and hypothesis are presented in Section 5. The hypothesis is tested through the experiments described in Section 6, the results of which and ensuing discussions being presented in section Section 7. Finally, concluding remarks and suggestions for future research are given in Sections 8 and 9, respectively.

2 Background

2.1 On the term *binarization*

Let us first disambiguate and delimit the use of the term *binarize*, as well as its various lemmatic variants, such as *binariz*[*ed*][*ation*][*ing*].

[1]The code can be found at `https://github.com/benluks/binASR`

For the purposes of this work, *binarization* refers broadly to approaches to deep learning that use networks whose *weights*, *biases* (if biases are used), and, in some cases, *activations* are constrained to the values -1 and 1, henceforth to be denoted as $\{-1, 1\}$ and ± 1. The choice of -1, as opposed to 0, for the alternate to 1 may seem unintuitive, seeing as binary operations between the values $\{0, 1\}$ constitute the fundamental building blocks of computers. One would therefore expect 0 for neural network binarization, rather than 1. As is demonstrated in this section, operations between values constrained to {-1, 1} can be effectively computed using hardware built on {0, 1}.

2.2 Basic hardware implications

While BNN implementations often rely on custom, task-specific hardware (Kim et al., 2014; Ardakani et al., 2018), an understanding of neural networks, the binarization approach at hand, and broad CPU functionality can help the reader recognize the device-agnostic implications of BNN research on hardware implementations.

The aim of neural network binarization is absolute quantization of the parameters and operations of the network. Quantization in this case refers to a reduction in numerical precision, as expressed in *bits. Absolute* quantization therefore refers to limiting said numerical precision to the smallest available unit of numerical distinction, i.e. a single *bit.*

2.2.1 Neural Network Basics

A neural network can be though of as a mathematical formula which computes a set of values based on an input. The output is determined by a set of parameters. For simplicity's sake we will limit these parameters to (1) scaling parameters, otherwise known as *weights* (2) biasing parameters, otherwise known as *biases*. In the interest of brevity, we consider a network with only weights for now. Consider a network that accepts 2 features as an input X and computes 2 features as output, Y, represented in Figure 1[2].

All features and parameters including inputs x_i, weights w_{ij}, and the output y_j for $0 \leq i < 2$ and $0 \leq j < 2$ are numerical values, generally floating points. Values y_j of the output vector $Y = \{y_0, y_1\}$ are computed by taking each input value x_i of the input vector $X = \{x_0, x_1\}$, multiplying it by its respective w_{ij}, and summing the values of said operation to obtain the output y_j. This operation is described formally in Equation 1 for any neural network with n input features:

$$y_j = \sum_{i=0}^{n-1} x_i w_{ij} \tag{1}$$

This operation can be generalized to networks with any number of input or output fea-

[2]this figure was adapted from `https://tex.stackexchange.com/questions/104334/tikz-diagram-of-a-perceptron#answer-104376`

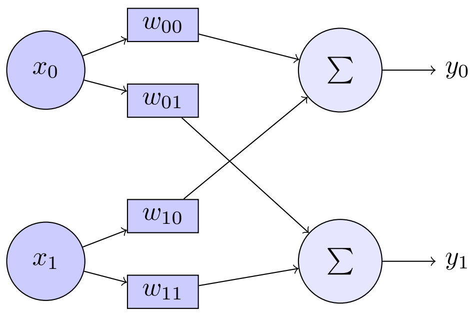


Figure 1: Diagram of neural network with n input features $x_{0,\ldots,n}$ and 1 output feature y. For simplicity, biases have been omitted.

tures by adjusting i and j to equal the number of input and output features, respectively. Furthermore, this operation can apply to networks with multiple layers. Subsequent layers accept the previous layer's output as input features, or x in Equation 1.

This series of multiplication and summation can be represented by matrix multiplications. In the case of a neural network layer with m input features and n output features, the weights W can be represented as a $m \times n$ matrix:

$$W = \begin{bmatrix} \mathrm{w}_{00} & \mathrm{w}_{01} & \ldots & \mathrm{w}_{0n} \\ \mathrm{w}_{10} & \mathrm{w}_{11} & \ldots & \mathrm{w}_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{w}_{m0} & \mathrm{w}_{m1} & \ldots & \mathrm{w}_{mn} \end{bmatrix} \tag{2}$$

such that each column represents the values i and each column the possible values for j in Equation 1. The value for an output feature y_j is therefore the sum of the element-wise product of the vector X and the j^{th} column of W. Using $\times$ to represent matrix multiplication, this can be expressed as:

$$y_j = \begin{bmatrix} \mathrm{x}_0 & \mathrm{x}_1 & \ldots & \mathrm{x}_m \end{bmatrix} \times \begin{bmatrix} \mathrm{w}_{0j} \\ \mathrm{w}_{1j} \\ \vdots \\ \mathrm{w}_{mj} \end{bmatrix} \tag{3}$$

2.2.2 Matrix multiplications, bit operations, and BNNs

Returning now to hardware implementations, let us consider the properties of BNNs.

Assuming that a BNNs' activations in addition to its weights are constrained to {-1, 1}, then the element-wise multiplications in Equation 3 are limited to:

Inputs		Output
A	B	A XNOR B
0	0	1
1	0	0
0	1	0
1	1	1

Table 1: *XNOR* logic gate functionality

$$\begin{cases} -1 \cdot -1 & = \ \ 1 \\ -1 \cdot 1 & = -1 \\ 1 \cdot -1 & = -1 \\ 1 \cdot 1 & = \ \ 1 \end{cases}$$

In other words, the element-wise multiplication is simply some function that returns a positive value when the two inputs match, and a negative one otherwise. Such operations are possible with the *XNOR* logic gate (Table 1).

By mapping -1 values to 0 and 1 to 1, the element-wise product of two vectors can be computed between two vectors, in this case some set of input features X having m elements, and some column $w_j = \{w_0 j, \ldots, w_{mj}\}$ of a weight matrix W, occupying a single bit per value x_i and w_{ij}. Considering that the same operations in full-precision networks use 32 or 64-bit floating-point arithmetic, the memory reductions of BNNs are significant.

The previous explanation assumes inputs or activations constrained to ± 1. However for numerous, specifically NLP-based tasks, the inputs are one-hot encoded vectors, such as in Hou et al. (2016); Ardakani et al. (2018); Liu et al. (2018). The element-wise multiplication in this case is considerably simpler: return row i of weight matrix matrix W such that input feature x_i of X is equal to 1. Recall that indexing is of constant ($O(1)$) time complexity.

This explanation is a massively simplified glance into the hardware motivations of BNN operations. It is naive, neglecting biases, matrix multiplication summation, and the possibility of more complex input features. This explanation does, however, demonstrate that (1) weight matrices can be stored effectively as binary strings in as many bits as it contains values, and (2) these binary strings need not be expanded to occupy greater space in order to undergo standard neural network operations.

3 Related Works

This section explores the individual avenues of research motivating the present work. These include binarized neural networks, BNN-based ASR, end-to-end ASR, and binarized recurrent units.

3.1 Binarized Neural Networks

The first experiment performed on a neural network with parameters constrained to $\{-1, 1\}$ appears in Gardner and Derrida (1988). In the interest of optimizing networks' sizes on limited computational storage, the authors evaluate the effects on network accuracy as numerical precision of said network's parameters are reduced. The authors demonstrate that, unsurprisingly, greater precision is preferable. It is, however, concluded that the viability of a given reduction factor, (i.e. the ability for a network to perform within reason) is task-dependent. Such a conclusion suggests that monumental reductions in numerical precision can potentially be applied to learning tasks, limited only by the formulation of the learning criteria. Subsequent research, to be discussed in the following section, would leverage error criteria and optimization algorithms, in efforts to reduce numerical precision with minimal accuracy reduction.

The back-propagation algorithm, (Rumelhart et al., 1986), a standard operation in present deep-learning algorithms, allows for small, incremental parameter updates, enabling nuanced representation learning. The success of this, however, relies on a numerical precision fine enough to allow for such small incremental shifts. The single-bit, all-or-nothing nature of BNN parameters therefore requires adjustments to prevent both stagnant parameters and overly-active parameter changes. Efforts to combat this issue relied on learning approaches deviating from back-propagation (Köhler et al., 1990; Golea and Marchand, 1993; Solla and Winther, 1998). The dominance of back-propagation based learning in more recent years has limited the applicability of other learning algorithms.

3.1.1 Straight Through Estimator

A solution enabling both parameter binarization and learning through back-propagation is the straight-through estimator (STE) (Bengio et al., 2013). A network's output, or *prediction* is measured against the intended outcome, or *target*, by some metric, otherwise known as the *loss function* or *criterion* for correctness. This correctness is expressed in the inverse as the prediction's *error* or *loss*. This loss is expressed at a single, real number. Training a neural network is the act of minimizing the loss on a given dataset. Back-propagation is therefore a calculation whereby the parameters of the network are measured in terms of their contribution to the loss. Very simply put, weights which contribute to an output which is ascribed a high loss are penalized proportionally to their contribution to said loss. The penalty is expressed in terms of a *gradient*. The gradient of a given parameter can be thought of as the parameter's contribution to the model's incorrectness. It, too, is a single value, and signals the extent to which the parameter should be adjusted in pursuit of a more accurate model. Let us walk through this explanation one more time, in order to better understand the connection between a model's performance and the trainability of its parameters:

1. The **loss** is computed in order to quantify the extent to which a model's output is incorrect.
2. A **gradient** is calculated with respect to said loss. This ensures that parameters are updated in proportion with the extent to which they are "responsible" for the loss.
3. Parameter updates are performed using this gradient, a detailed explanation of which can be found in Trehan (2020).

The STE is an innovation that enables us to leverage the sophistication of back-propagation when training BNNs. Consider in a full-precision neural network some weight $w = 0.350$. Say now that the gradient computed after the forward pass and loss calculation is $\nabla = 0.844$. The parameter update step involves subtracting (or adding, depending on the choice of notation) the gradient from the parameter, often in the order of $\nabla w \cdot 10^{-3}$. The latter expression is the *learning rate*, intended to inhibit volatility in parameter changes to enable gradual updates. The so-called *optimizer step* and resulting weight w', in this case, would be:

$$w' = 0.350 - 8.44 \times 10^{-4} = 0.296 \tag{4}$$

This subtle, incremental updates is crucial to the back-propagation algorithm and its ability to nudge the network towards better performance (Rumelhart et al., 1986).

Weights in BNNs constrained to ± 1 cannot be subject to the small, incremental updates necessary in order to undergo gradient descent. Binarized parameters only have the ability to stagnate or reverse, which begs the question of how to handle real-number updates. Are they rounded to the nearest values in $\{0, 1\}$ or $\{0, 2\}$? If they are as small as in Equation 4, then one can expect no parameter updates, and an ever-stagnating performance. In contrast, if the learning rate is set to an abnormally large value, then the network will update drastically with no improvement to the performance (Zeiler, 2012).

The innovation of STE is to perform parameter updates on a set of full-precision parameters, binarizing the parameters on the forward pass. This means that an output is computed using parameters constrained to ± 1. The loss is then computed against the model's output. The gradient, too, is calculated with respect to the real-valued weights. In the optimizer step, however, the full-precision weights are updated. In the subsequent forward pass, the full-precision weights are binarized, their values exercising influence over the values of the resultant binarized weights. This has the advantaging of leveraging slow updates for consistent convergence on a set of parameters whose values do not lend to small, incremental change (Bengio et al., 2013; Xiang et al., 2017; Simons and Lee, 2019).

3.1.2 BinaryConnect

The ability to adapt standard training protocol in order to train and test BNNs has enabled researchers to investigate the applicability of the technique on numerous existing neural network architectures. The first work of this sort uses a method referred to as BinaryConnect (Courbariaux et al., 2015), which achieves near state-of-the-art performance on various image classification tasks. BinaryConnect is the basis of much–arguably most of the BNN research to succeed it (Hubara et al., 2016; Courbariaux et al., 2016; Rastegari et al., 2016; Simons and Lee, 2019; Mishra et al., 2020; Yuan and Agaian, 2021). This section discusses the BinaryConnect method, a few of its notable successors, and motivates the need for innovation in order to be applied to new tasks, namely, sequence-to-sequence based ones.

The authors of BinaryConnect introduce two options for calculating the binarized weights during the forward pass, namely (1) *deterministic* and (2) *stochastic*:

1. **Deterministic** binarization is a simple matter of binarizing according to the sign function. In other words, each weight w_{ij} in a weight matrix W is binarized according to (with notation taken from Courbariaux et al. (2015):

$$w_{b_{ij}} = \begin{cases} +1 & \text{if } w_{ij} \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

   Note that $w_{b_{ij}}$ in this case refers to the binarized version of weight w_{ij}.

2. **Stochastic** binarization computes the value of $w_{b_{ij}}$ probablistically. In the interest of simplicity and generality, this explanation of stochastic binarization is somewhat less detailed than that offered in Courbariaux et al. (2015). Some function f is used to fix the values of W between 0 and 1, greater values w_{ij} being closer to 1 and vice-versa. The fixed values are then treated as probabilities. The greater the probability, the greater chance of $w_{b_{ij}}$ being $+1$, and the lower the probability, the greater chance of it being -1. These outcomes are drawn from a uniform distribution. Per Courbariaux et al. (2015)'s notation:

$$w_{b_{ij}} = \begin{cases} +1 & \text{with probability } p = f(w_{ij}) \\ -1 & \text{with probability } 1 - p \end{cases}$$

   In the case of BinaryConnect, f is a so-called *hard sigmoid* function $\sigma(x)$, wherein 1 is added to the real-valued weight w_{ij}, the result is divided by 2, and the result of that is clipped at $\{0, 1\}$:

$$\sigma(x) = clip(\frac{x+1}{2}, 0, 1)$$

   Stochastic binarization can be seen as treating the f-treated weights as a Bernoulli distribution adjusted for the ± 1 constraint of binarization. In this method, the full-precision value w_{ij} does not *determine* $w_{b_{ij}}$, but rather influences it.

The authors attribute the success of BinaryConnect to its utilizing binarization as a means of regularization. Regularization in this case refers to techniques that reduce overfitting the network to the training data (Kukačka et al., 2017). More specifically, in this case binarization introduces noise, i.e. it obscures the precision of the network information in order to reduce overfitting, much in the same spirit as dropout (Srivastava et al., 2014).

Prominent successors to BinaryConnect include BinaryNet (Courbariaux et al., 2016) and XNOR-Net (Rastegari et al., 2016). BinaryNet is an extension of BinaryConnect that binarizes not only the network's parameters, but also the activations. This enables the network to use bit operations in computing all layers. XNOR-Net improves upon this implementation by adjusting the binarized parameters with the use of a scaling factor. This parameter adjustment is implemented with the theoretical goal of regaining some precision lost in the binarization process (Simons and Lee, 2019). XNOR-Net outperforms its BNN predecessors on the ImageNet (Deng et al., 2009) image classification task. This also presents the first successful application of BNNs to large-scale datasets, a weakness of BNNs that Courbariaux et al. (2015) and Courbariaux et al. (2016) acknowledge.

Below, I summarize three crucial takeaways from this section:

1. **Large deep neural networks contain a certain degree of redundancy which can be exploited through the reduction of numerical precision.** Reducing the precision of the network's parameters by a factor of 32 (namely from 32-bit floating point precision to 1-bit binary values) results in a comparatively modest reduction in accuracy. As will be discussed below, strategic implementational choices help to regain some of this accuracy, which is a further testament to the relative superfluity of such precision.

2. **Regularization/noise is a delicate balance.** Courbariaux et al. (2015) welcome the noise brought from binarization as a means of regularization. The extremity of this noise does, however, negatively impact the network's performance, as well as its ability to generalize to larger datasets.

3. **Precision can be artificially regained with little compromise to performance.** As Rastegari et al. (2016) demonstrate, the forfeiture of accuracy can be compensated for with operations implemented on the network's parameters, rather than hampering the compression of the network. This is promising, as it demonstrates, per the first bullet-point, how simple changes to BNN architectures can improve the network's performance without sacrificing compression.

3.2 BNNs for ASR

Note that the models discussed in Section 3.1.2 refer only to computer vision tasks. Early work in BNNs focused largely on image classification and similar tasks (Simons and Lee, 2019; Yuan and Agaian, 2021). Nonetheless, a number of efforts have been made to apply highly-quantized networks to ASR (Kim et al., 2014; Xiang et al., 2017; Qian and Xiang, 2019; Gao et al., 2021). Kim et al. (2014); Xiang et al. (2017); Qian and Xiang (2019) use hybrid, HMM-DNN models, whereas Gao et al. (2021) uses a convolutional model. The approaches are discussed below.

Kim et al. (2014) perform ASR with networks whose weights are constrained to $\{-1, 0, 1\}$. Although some inconsistencies exist in the literature regarding whether or not to include these "ternary" networks in discussions of BNN, such networks are not considered in this work. Nonetheless, the work of Kim et al. (2014) is worth mentioning as an early proof-of-concept of the viability of very low precision networks for ASR tasks. Its relevance in this case is limited, however. It predates STE, and therefore relies on training methods of much narrower applicability than back-propagation based ones.

Xiang et al. (2017) and Qian and Xiang (2019) apply the approach introduced in Courbariaux et al. (2015) to hybrid ASR. Xiang et al. (2017) apply BNNs to phone recognition tasks in HMM-DNN networks. Per the discussion on hardware implementations in Section 2.2.2, the first layer's weights are not binarized, but rather are computed in a full-precision matrix multiplication with the network inputs. According to the authors, speech features do not lend themselves to binarization. The results show only a minor loss in accuracy compared to the full-precision model.

Qian and Xiang (2019) extend the approach of Xiang et al. (2017) with a more complex neural network. Numerous experiments are conducted with a combination of convolutional and feed-forward layers. In the experiments, various subsets of the network's layers, (i.e. only the convolutional layers, only the feed-forward layers, and a combination of both) are binarized and compared for their effect on performance. Results show that binarizing only convolutional layers causes a negligible increase to the word-error-rate (WER). Such an observation is consistent with the earlier insights indicating the effectiveness of binarization in computer-vision tasks, on CNN-based networks (Courbariaux et al., 2015; Rastegari et al., 2016). This is of course of limited utility, as such an approach preserves many of the full-precision weights. This paper also shows that the accuracy of a BNN can be imrpoved through *knowledge distillation* (Hinton et al., 2015) from its full-precision counterpart. This is a common technique for boosting the performance of small-footprint networks (Howard et al., 2017) and could likely be of use for BNN research, but it is beyond the scope of this discussion.

Following the success of binarization on convolutional units, Gao et al. (2021) introduce 1-bit WaveNet. This work constructs a network based on WaveNet (van den Oord et al., 2016). WaveNet operates on raw audio waveforms, using multiple layers of convolutions on the signal, much like in computer vision, in order to distill the raw audio into features relevant to the task. It should be noted the the original WaveNet was mainly applied to generative tasks (i.e. creating more audio), rather than classification or recognition. Gao et al. (2021) apply this model to keyword recognition tasks. This approach is clever, as it leverages the success and original application of BNNs, namely convolutional units, and transfers it to the audio domain on an already performant model. Nonetheless, the purely convolutional WaveNet architecture is idiosyncratic, never having achieved popularity in the speech technology domain. There is therefore little-to-no scholarly basis supporting its ability to perform large-vocabulary ASR tasks. For this reason, no further discussions on the matter will ensue.

3.3 End-to-End Speech Recognition

End-to-end speech recognition refers to approaches to ASR which yields transcription predictions given acoustic speech features. What exactly qualifies as *end-to-end* in an environment where an increasing number of components are being performed in the acoustic model itself, such as feature extraction (Schneider et al., 2019; Sainath et al., 2013; Palaz et al., 2013) and language modelling (He et al., 2018). For the purposes of this work, models which accept as input speech features or raw audio, and output some transcription, be it phonetic, character, word-piece, or word-based, are considered end-to-end.

The concept exists so as to be distinguished from the predecessor and long-time dominant approach to ASR: Hidden Markov Model (HMM)-based speech recognition Graves (2006); Graves et al. (2013); Graves and Jaitly (2014). HMM-based recognition is broken down into a complex pipeline of components, including duration modelling, phonetic transcription, HMM-state training, possibly acoustic modelling with Gaussian Mixtures (GMMs), etc. (Rupali et al., 2013). Improving such systems therefore requires diagnostics on individual components, changing them individually before testing their improvement with respect to the entire system Graves et al. (2013). End-to-end systems are therefore advantageous, as these unique tasks, (i.e. duration modelling, acoustic modeling, etc.) are performed implicitly in the network, capable of updating in tandem to and in accordance with one-another.

3.3.1 Connectionist Temporal Classification

Graves (2006) is an early approach to forego HMMs in favour of neural networks in an ASR pipeline. This method introduces Connectionist Temporal Classification (CTC), allowing a model to learn alignments between speech features and the transcribed characters, a separate component in HMM-based ASR. The system uses Recurrent Neural Networks (RNNs), capable of factoring temporal information in the interest of sequence modelling. RNN-based models would come to dominate ASR, and are the focus of the present work.

End-to-end ASR with CTC-based models has shown excellent performance, with notable examples including Graves et al. (2013); Hannun et al. (2014); Amodei et al. (2015). Some of these models leverage the performance improvement of Long-Short-Term-Memory (LSTM) (Hochreiter and Schmidhuber, 1997) or Gated-Recurrent-Unit (GRU) (Cho et al., 2014) cells. These cells replace, or rather upgrade the funtionality of so-called "vanilla" RNN cells. These improvements have become the standard unit in recurrent models. The term "RNN" is henceforth used as an term to denote a model built with some variant thereof, either LSTM or GRU.

CTC-based ASR operates as follows: A network accepts some input X of length T. At each time step $t \leq T$, a vector of speech features x_t is passed through the network, containing at least one recurrent (i.e. RNN) layer. The output of this forward pass is a prediction over the set of possible character transcriptions, generally represented as *log-probabilities*. The output y_t at time t would therefore be something to the effect of:

$$y_t = \{-3.884, -1.030, \ldots, -4.279\}$$

the indices of which denoting a probability of a given character in a set of characters $chars$, e.g.:

$$chars = \{< blank >, < space >, < apostrophe >, a, \ldots, z\}$$

such that the lengths of y_t and $chars$ are equal.

Note the first character in $chars$: $< blank >$. This character is the key innovation of CTC-based ASR. Considering the mismatch between the length of speech features and the characters in its transcription, there must be some way for the network to express that, at a given input x_t, no new characters are being predicted, i.e. the character predicted at x_{t-1} is still being uttered at x_t, or there is a break before the following one.

In decoding a CTC output, repeated characters are removed, and then $< blank >$ characters are removed. Therefore, a network might output a prediction $\hat{y}$ such as, ($-$ representing the $< blank >$ character):

$$\begin{aligned}\hat{y} &= hhhh - - - ee - - lll - - l - - oo - - \\ &= h - e - l - l - o \\ &= hello\end{aligned}$$

Note the "$-$" separating instances of the letter "l", indicating the presence of 2 separate "l" characters. The CTC loss function remains agnostic to the particular $X - Y$ alignment, and computes the error with respect to any possible alignment of the transcription.

The primary drawback which authors note about CTC-based ASR is the absence of "language awareness" (Bahdanau et al., 2014). Each output y_t depends on x_t and surrounding hidden activations, but not on surrounding outputs $\{y_u : u \neq t\}$. This is a problem, as there is no one-to-one mapping between English phonemes and graphemes. The output y_t relies on information from surrounding outputs. An obvious example of this would be the decoding of speech features resembling the phoneme /f/, and determining whether to transcribe them as *f*, *ff*, *gh*, or *ph*.

In order to solve this, CTC decoding tends to rely on beam-search decoding (Scheidl et al., 2018) with a language model. This algorithm compares the network's ascribed likelihoods of certain outputs with the language model's computed likelihood of certain outputs. Rather than selecting only the top character at each output step y_t, the k most likely sequences are kept, step-by-step, the most likely of which (as determined by some weighted comparison between the language model and the acoustic model) is returned. k is a hyperparameter referred to as the "beam-width". The influence of the language model in computing the combined likelihood is controlled through another hyperparameter, refered to as the "language model weight", w_{lm}. If w_{lm} is too high, the output will disregard the acoustic features and return high probability sequences from the language model. If it is too low, the output will be gramatically incoherent, confusing like-sounds and homophones. It goes without saying that the necessity of added components, in this case the language model, further obscures the goal of "end-to-end" speech recognition, and still necessitates the configuration of separate components.

3.3.2 Encoder-Decoder

Another effective option in RNN-based end-to-end ASR has been the *encoder-decoder* architecture (Sutskever et al., 2014; Cho et al., 2014). Encoder-decoder architectures are used for tasks that map sequential inputs to sequential outputs. This architecture, originally applied to neural machine translation (Sutskever et al., 2014), takes as input a sequence $X = \{x_0, \ldots, x_T\}$ of length T. The main components of the architecture, per its name, are (1) the *encoder* and (2) the *decoder*. The network functions as follows:

1. *Encoding*: X is passed through the encoder, an RNN, to obtain a hidden state h_T. This can be thought of the hidden state at time t. h_T is a fixed-length vector, meaning that its size is independent of the length T of X, but rather determined by a hyperparameter of the encoder RNN, namely the network's *hidden size/dimension*.

2. *Decoding*: h_t is treated as an initial hidden state for the decoder network. The decoder network, also an RNN, accepts as input the previous hidden state (beginning with h_t) and the previously generated token (beginning with a "start" token), and returns (a) a probability distribution predicting the next generated token y_u at decode time step u, and (b) the following hidden state $h_{dec,u}$ to be used as input to the upcoming decode step $u + 1$. The decode process repeats until some "end" token is generated, or u has reached some maximum length hyperparameter, established in order to keep the decoding process from repeating ad infinitum.

In the case of ASR, the X can be thought of as speech features, such as mel spectrogram slices, t being the horizontal length of said spectrogram. $Y = y_0, \ldots, y_u$, in this case, would be the token predictions, being phonemes, characters, words, or word-pieces.

3.3.3 Encoder-Decoder + Attention

The "vanilla" encoder-decoder architecture has performed rather unimpressively in ASR tasks (Lu et al., 2015), with results inferior to those of traditional, HMM-based systems. performance improvements would come later, with the innovation of attention mechanisms (Graves et al., 2013; Bahdanau et al., 2014).

Attention-equipped RNN encoder-decoders have lent significant contributions to the improvement of End-to-end ASR. The so-called *attention*-mechanism was first used in Graves et al. (2013) for handwriting synthesis, being applied shortly thereafter the neural machine translation (Bahdanau et al., 2014). Chorowski et al. (2014) is the first use of such a mechanism for ASR. While early performances using this mechanism offer, at best, comparable performance to hybrid (i.e. HMM-DNN-based) ASR (Chorowski et al., 2014, 2015), later improvements to this architecture offer state-of-the-art performance (Chan et al., 2015; Chiu et al., 2018).

The attention mechanism can be seen as a component in an RNN encoder-decoder, between the encoder and the decoder. It improves performance by allowing the decoder to "focus on" particular portions of the input sequence when making token predictions.

During encoding, at each time-step t, the encoder emits and stores an encoding h_t. After encoding, the matrix H of length T is a sequence of vectors, each of which representing the inputs at their respective time-step t. During decoding, the attention mechanism uses the decoder hidden state $h_{dec,u-1}$ along with the encoder hidden states H to produce an attention vector a of length T, summing to 1. This vector is treated as a weighting over H. The vectors h_t of H are summed according to their respective weightings, and passed along with the previous token prediction y_{u-1} into the decoder RNN in order to produce the next prediction y_u and decoder hidden state $h_{dec,u}$.

This weighted sum, sometimes referred to as the *context*, can be seen as a prediction about which input features, and in what proportions, are relevant to predicting the upcoming token. Figure 2, from Bahdanau et al. (2014) shows the learned attention vectors in predicting a French sequence from an English input in neural machine translation, lighter squares representing greater values, and vice-versa. This figure should be read as follows:

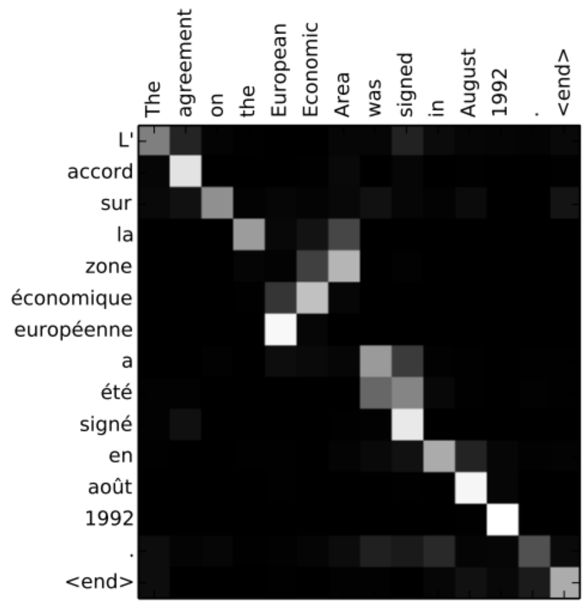


Figure 2: Visualization of attention vectors in English-to-French neural machine translation task, from Bahdanau et al. (2014)

1. The whole input (English) sequence is visible. The model "looks at" the entire English sequence, and returns the weightings a over the English tokens. In this case, *The* is deemed to be, by far, the most relevant to generating the first output, i.e. the first French token.

2. Using this attention vector a, a weighted sum over the inputs is computed and used to predict the first token. This weighted sum is expressed as the first row in Figure 2. In this case the prediction is *L'*, the French definite article, or direct

translation of English *the*. Note the some energy is given to the second English word as well, *agreement*. This information is also pertinent to the model, to a lesser extent. The following French word, *accord*, begins with a vowel, thus influencing the choice of article (*L'* rather than *Le*).

3. Having made the first prediction, the following attention vector is computed over the English sequence, expressed as the second row. This, along with the inputs and previous token prediction are used to make subsequent predictions, and so forth. This continues until the *<end>* token is predicted.

There are a few points of particular interest in Figure 2. For one, note that the fourth predicted token, *la* (en: *the*, feminine) focuses on both *the* and *Area*. This is because the choice of definite article relies on the gender of the noun it references. This signals the model to choose a feminine article, as it modifies *Area*, which the model predicts as having, in this context, some feminine translation.

Another point of interest is a visual representation of the mismatch of order between the English *European Economic Area* and French *zone économique eurpéenne*. Each maps to an individual word in the other language, but in a separate order, and the attention mechanism leverages both a knowledge of syntax as well as word-for-word translation in predicting this.

Last and most interesting is the way the mechanism handles a mismatch in number of words to express a similar construction. The preteritive *was* is translated to the perfective. In this case, both components of the French *a été*, a morpho-sytactic equivalent of *has been*, "attend" to the entire verb complex *was signed* in order to capture and convey the proper aspective meaning of the English. This is an example of forfeiting literal, word-for-word translation in favour of capturing a more accurate meaning.

How attention applies to ASR is somewhat more intuitive. Each encoder hidden state h_t represents a slice of the input features. Figure 3 from Chan et al. (2015) illustrates the attention vectors computed over input features that factor into subsequent input predictions. One can note from Figure 3 how attention can spread focus across a small region of an utterance in order to factor coarticulation and surrounding sounds into token predictions.

A deviation from neural machine translation is the monotonic (i.e. left-to-right) constraint on the attention focus. In neural machine translation, each decoding step y_t can rely on any combination of input values. Speakers of multiple languages can attest to the lack of uniform word order across languages. As can be noted in Figure 3, the attention mechanism focuses on the portion of audio that is most relevant to the character, word, or word-piece that is being decoded, which includes segments in the same place as– or to the right of the previous attention focus. Chorowski et al. (2014) account for this by limiting the available input steps x_t on which the attention mechanism can focus to a small, monotonically moving window. The authors show that limiting the search space to a subset of X decreases memory usage, speeds up convergence, and improves overall performance.

Attention-based architectures have achieved success in ASR tasks, spurring numerous

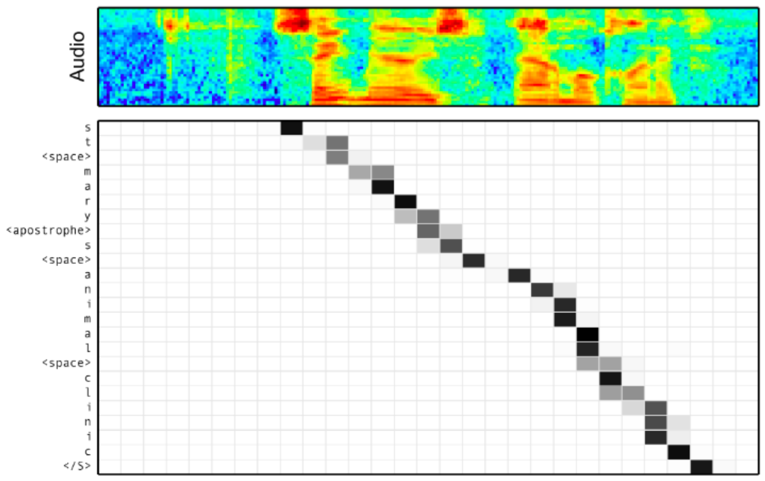


Figure 3: Visualization of attention vectors in a speech-to-text task, taken from Chan et al. (2015)

variants (Bahdanau et al., 2016; Kim et al., 2017; Watanabe et al., 2017; Zhang et al., 2017; Raffel et al., 2017; Chiu et al., 2018; Zeyer et al., 2018; Bello et al., 2019). Bahdanau et al. (2016) use time downsampling in the encoder. Watanabe et al. (2017) incorporate CNN modules in the encoder, whereas Zhang et al. (2017) introduce LSTM cells whose gates are computed using convolutional, rather than simple linear operations. Kim et al. (2017) use a loss function that combines a standard log-likelihood-based classification criterion with the CTC loss in order to leverage the contextual awareness of attention with the noise robustness and forced directionality. In terms of forcing directionality, Raffel et al. (2017) implement forced monotonic alignment, i.e. attention that centers its focus only on the right of the attention focus of the previous decoder step. This reduces training and inference time, as a smaller window of values is computed in the "attending" phase of the forward pass. Such a strategy appears as early as Chorowski et al. (2014), but is calculated algorithmically, whereas the window size and location of the attention focus is learned as a parameter of the network in (Raffel et al., 2017).

3.4 Binarizing Recurrent Units

In spite of the relative popularity of STE-based BNN research applied to convolutional and feed-forward neural networks, there exist comparatively few efforts to apply these techinques to RNN-based networks. These works are detailed in this section.

The first effort to binarize recurrent units is found in Ott et al. (2016). The authors perform basic character-level language modelling and digit recognition experiments on RNNs and its variants, namely, GRUs and LSTMs. The networks are subject to binarization and ternarization. Both stochastic and deterministic binarization are performed using the BinaryConnect approach discussed in Section 3.1.2. The authors achieve near-chance results on the aforementioned tasks and conclude, therefore, that binarization does not lend to RNNs. Note that the authors attempt only to binarize GRUs, which have generally been shown to perform as well as LSTMs Cho et al. (2014) despite the former containing fewer parameters.

Ott et al. (2016) explain the poor performance as follows: RNNs by nature re-use the same set of weights W for each time-step x_t in the input sequence X. This relatively small number of weight parameters fails to capture the subtlety of the training data, leading to high variance, or overfitting. This is in part the motivation behind the advent of LSTMs (Hochreiter and Schmidhuber, 1997), which increase the number of parameters in order to selectively control the flow of information across the network over time. Because of the scarcity of parameters in W, RNNs rely on the granularity afforded by $32/64$-bit numerical precision in order to capture details across a variety of data. One can think of each weight w as scaling the proportionality of correlation between a given input feature to a given output feature. The key to a network's ability to generalize to a dataset therefore relies on capturing the relevance of input features without exaggerating their importance or else said feature's presence will disproportionately influence the network's output, hence, overfitting. The problem with binarization, therefore, is that near-zero valued weights on which the network relies are forced to large values, namely 1 and -1, which in turn "emphasizes" subtleties in the data that the original, full-precision, near-zero value would otherwise merely "acknowledge". Subsequent works seek to overcome this.

Edel and Köppe (2016) apply binarized bidirectional LSTMs to human activity recognition with considerable success. Compared to other approaches to such task, binarized LSTMs fall short only of their full precision counterparts. Following these results, the authors speculate that for certain sequential tasks, capturing temporal dependencies (as RNNs do) is more important than the precision of the parameters themselves, as the binarized LSTMs performance suggests, in comparison to a full-precision feed-forward network. When comparing the results in Edel and Köppe (2016) with those from Ott et al. (2016) using GRUs, one could reason that the abundance of parameters in LSTMs, considered redundant in full-precision networks (Cho et al., 2014), may compensate for the loss of precision resulting from binarization.

Hou et al. (2016) consider various methods of optimizing binarized parameters. In this method, which the authors refer to as *Loss-Aware Binarization*, the effects of binarization are considered and minimized as a separate loss term. This is done in an attempt to solve the problem that, when binarized, deep recurrent neural netowrks quickly suffer from exploding gradients (Bengio et al., 1994). This is in contrast to earlier approaches, which binarize parameters and compute the loss, but do not consider effect of binarization on the loss, i.e. how much less accurately a BNN performs, relative to its full-precision counterpart, at each forward pass. This network outperforms existing binarization approaches on character-level language modelling tasks.

Liu et al. (2018) apply a single-layer LSTM language model with binarized input and output embedding layers to various NLP and ASR rescoring tasks. Their results demonstrate the viability of binarization of the linear layers in LSTMs. Furthermore, they are the first to apply this technique to word-level language modelling. Beyond this, the relevance of their research is of limited relevance to the present work, as the effects binarization on LSTM depth is not studied.

In a similar vein, Xu et al. (2018) search for optimal quantization parameters with the use of a binary search tree. I use *quantization* here as opposed to *binarization*, as this paper is not specific to binary, i.e. 1-bit parameters. Instead, the authors develop a method that applies to k-bit quantization for any value k. Similar to Hou et al. (2016), the quantized weights W_b are not determined through a fixed equation determined (or in the case of stochastic quantization, influenced) only by the full-precision weights W. Instead, the possible quantization values, normally $\{-1, 1\}$ are scaled by some factor α, referred to as a *code*. The optimal value of α is learned in the training phase.

Experiments with this method are performed on character-level language modelling with reasonable success. While the results are promising for relatively low values of k, the decrease in performance when $k = 2$ implies that this method does not lend itself to the extreme quantization that BNNs undergo. Note that this method applies only to values of k such that $k \geq 2$, disqualifying it as a binarization method, strictly speaking. It is nonetheless discussed here for its being the first highly quantized model to perform word-level language modelling, and serving as inspiration to Ardakani et al. (2018), a work of immense importance to the present one.

3.4.1 Learning Recurrent Binary/Ternary Weights

The central focus and primary inspiration of the present work is the method of binarizing weights introduced in Ardakani et al. (2018). This paper is discussed here insofar as it inspired by and contributes to the literary cannon pertaining to binarized RNN research. An in-depth explanation of the technicalities of the authors' approach follows in Section 4.

Ardakani et al. (2018) introduce a method, to which the authors refer as Learning Recurrent Binary/Ternary Weights (henceforth LRBW). This aim of their research is to develop an approach that can be applied across a variety of sequential tasks. Hou et al. (2016) and Xu et al. (2018)'s works are only tested on single tasks, namely character-level and word-level language modelling, respectively. In terms of word-level language modelling, Xu et al. (2018) find that 4-bits are required to perform up to par with full-precision networks, 2-bit quantization causes a dramatic increase in error, and 1-bit, fixed-value binarization is not explored at all.

The authors introduce batch-normalization (Ioffe and Szegedy, 2015) to the computations of the gates within the LSTM cell (explained further in Section 4). The reason is as follows: LSTMs are particularly sensitive to unnormalized input (Cooijmans et al., 2016; Hou et al., 2019). Just as RNNs rely on subtlety in the individual weight values w_{ij} in order to control the flow of information, they also rely on distributions that are not too extreme. Batch normalization simultaneously adjusts the values to an optimal scale, and maintains a normal distribution of values, in order to avoid offsetting, or erasing the information passed from previous timesteps. In full-precision LSTMs, strategic weight-initialization prevents this (Glorot and Bengio, 2010). However, as BNN weights are dynamically changed to values more extreme than the initial weights, batch normalization is required.

Batch normalization is generally used to normalize inputs X in feed-forward and convolutional neural networks in an attempt to stabilize convergence. It should be noted, however, that it is considered ineffective in RNNs, due to the variable input length and shared weights across input steps (Ba et al., 2016). Since the output of an RNN h_t at time t (also called the hidden state) is used to compute the hidden state at the following time step h_{t+1}, the use of batch normalization essentialy equates the hidden state h_t as inputs X to the following time step $t+1$. This is somewhat analogous to the layer-wise forward pass in feed-forward neural networks, wherein a layer l's output a^l relies on the previous layer's output a^{l-1}. In keeping with RNN tradition, however, batch normalization is not applied to the inputs X.

In addition to computing the mean and variance, batch normalization includes learnable scaling and biasing parameters, ϕ and γ, respectively. Furthermore, batch normalization is not applied at inference time (ever, not just in this work). LRBW can therefore be categorized as an approach wherein the quantized parameters are learned, similar to Hou et al. (2016) and Xu et al. (2018), through binarization-specific parameters.

LRBW exhibits superior performance to previous aprroaches. Furthermore, the same model and quantization approach are used regardless of the task. These include character-level language modelling, word-level language modelling, and, notably, question answering per DeepMind's Attentive Reader (Hermann et al., 2015). This task takes a question and a context as input. The two are passed, word-by-word, through an RNN, after which an attention mechanism similar to those discussed in Section 3.3 attends to the input representation and generates an answer sequentially. Note the similarity between this model and encoder-decoder + attention ASR models.

4 Learning Recurrent Binary Weights

This section elaborates further on the work of Ardakani et al. (2018). The experiments that follow rely heavily on the LRBW approach from this paper. As a fully-realized implementation is to follow, this section explains the technicalities of this approach, including minor specifications that are important, should the reader choose to attempt to reproduce the experiments.

4.1 The anatomy of an LSTM cell

In order to understand the approach of Ardakani et al. (2018), let us first take a closer look at the operations that comprise a time-step in an LSTM cell. The following representation, notated as in Ardakani et al. (2018), considers the forward pass in an LSTM cell having weights W and no bias, at time-step t, given a set of input features x_t:

$$
\begin{aligned}
f_t &= \sigma(W_{fh}h_{t-1} + W_{fx}x_t) \\
i_t &= \sigma(W_{ih}h_{t-1} + W_{ix}x_t) \\
o_t &= \sigma(W_{oh}h_{t-1} + W_{ox}x_t) \\
g_t &= \tanh(W_{gh}h_{t-1} + W_{gx}x_t) \\
\\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
\tag{5}
$$

where σ and $tanh$ represent nonlinear activation functions, namely *sigmoid*, and *hyperbolic tangent*, respectively, and $\odot$ represents the hadamard, or element-wise product of two identically-shaped matrices.

The first four terms f_t, i_t, o_t, and g_t are the cell's gates. There are referred to as the cell's *forget*, *input*, *output*, *cell* gate, respectively. Each gate has 2 sets of weights. The first, per the notation above, is the *hidden-hidden* weight $W_{\{gate\}h}$. The other, $W_{\{gate\}x}$, is the *input-hidden weight*. The two sets of weights, much as in a standard forward pass, are used in a matrix multiplication with their respective inputs.

The purpose of these gates is control over the flow of information. As each gate has separate weights, the gates can compute values that are different from one another given the same set of inputs x_t and h_{t-1}. As these values are combined with one another through various operations, the forward pass through an LSTM cell is afforded more precise control over which information is passed from cell-to-cell and to subsequent layers. How this works, more precisely, is outlined in the following paragraph. Note that "information" is used here to speak in the abstract of the real-number vectors x, c, and h, whose values qualify the network's "learning".

Starting from the bottom, let us consider how the values of these cells control the flow of information. h_t is the cell's hidden state, treated as both an output of the cell (final or passed to subsequent layers) and a state passed to the following time step, represented in Equation 5 as h_{t-1}. Ignoring the nonlinearity function, h_t is a product of the cell-state c_t and the output gate. Considering the metaphorical use of *gate* here, one can think of o_t determining the scale at which certain information from the cell state is relevant to the output. In a sense, the relevance of each output feature at each time-step is scaled based on its relevance to the intended output. Figure 4 from Ardakani et al. (2018) is a histogram of the values of o_t over 100 time steps (vertical axis) on a character-level language modelling task. The figure demonstrates the the gate is rather selective, opting largely for near-zero values, showing that relatively few features, in few time-steps, are of remarkable importance to the network's output.

Preceding the hidden state is the cell state, c_t. First, the forget gate is multiplied with the previous cell state. This gives this cell the opportunity to forget information from previous time-steps. As an extreme example, if f_t were to contain only 0 values, outputs from time t onwards would cease to rely on information from before t. Next, the cell gate, g_t, is scaled according to the input gate i_t. g_t can be thought of as offering new

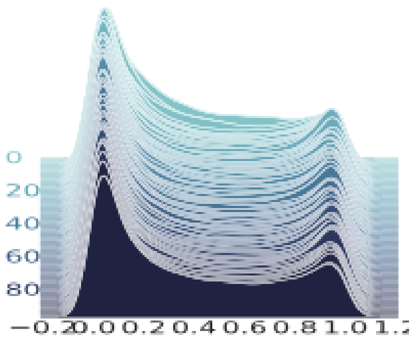


Figure 4: Histogram of output gate values on a character-level language modelling task.

information. i_t is then tasked with determining the importance of said information in computing subsequent states/outputs. By summing the two products, c_t becomes some strategic combination of previous information and new information.

The innovation of the cell state c_t contrasts with *vanilla* RNNs, wherein h_t serves as both the output and the only recurrent connection. Disentangling the recurrent connection from the cell's output acknowledges that the relevance of information to the current output h_t may not bear the same relevance to subsequent outputs $h_{t+n\geq 1}$.

Understanding the flow of information in the LSTM cell, one can better understand the purpose of having separate weight values for the respective gates. These weights essentially learn, through network training, how to best process input information in order to preserve, forget, and update the learned information over time.

4.2 Adding Batch Normalization

Let us now consider the addition of batch normalization, as detailed in Equation 6. Let $W_{\{gate\}\{h|x\}_B}$ denote the binarized version of the weight matrix $W_{\{gate\}\{h|x\}}$, for a given gate $\{gate\}$ and input/state $h|t$, and BN denote the batch normalization operation:

$$
\begin{aligned}
f_t &= \sigma\Big(BN(W_{fh_B} h_{t-1}, \phi_{fh}, 0) + BN(W_{fx_B} x_t, \phi_{fx}, 0)\Big) \\
i_t &= \sigma\Big(BN(W_{ih_B} h_{t-1}, \phi_{ih}, 0) + BN(W_{ix_B} x_t, \phi_{ix}, 0)\Big) \\
o_t &= \sigma\Big(BN(W_{oh_B} h_{t-1}, \phi_{oh}, 0) + BN(W_{ox_B} x_t, \phi_{ox} 0)\Big) \\
g_t &= \tanh\Big(BN(W_{gh_B} h_{t-1}, \phi_{gh}, 0) + BN(W_{gx_B} x_t, \phi_{gx}, 0)\Big) \\
\\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh\Big(BN(c_t, \phi_c, 0)\Big)
\end{aligned}
\tag{6}
$$

Observe that for each gate there exist two BN terms: one for the h_{t-1} and one for x_t. This brings the total of unique BNs, prior to considering the c_t, to 8– 9 with c_t. Each term contains its own scaling parameter $\phi_{\{gate\}\{h|x\}}$, which is learned during training. Although not explicitly established in the paper, I take these parameters to be the namesake of "learning" in the title of the paper and method. Note, also, the 0 in each $BN()$. This indicates that batch normalization does not contain biasing, consistent with the LSTMs in this method.

Equipped with our understanding of the operations in the basic LSTM cell, let us now consider how batch normalization alleviates the drawbacks of binarization. As mentioned earlier in Section 3.4.1, binarization forces the weights W to extreme values. This has a tendency to inflate the subsequent activations a. In other words, after passing the through the non-linearity, one could expect activations to tend much more closely to the extrema of their respective non-linearities, i.e. 0 and 1 for σ, -1, and 1 for tanh. Recall, per the discussion above, that the function of the LSTM's gates is to scale information relative to its relevance. The result of extreme-valued gate activations is that information is essentially categorically forgotten or dramatically over-emphasized. This will result, once again, in extreme values for outputs. Rather than incrementing towards an optimized output, as is the function of gradient descent based algorithms (Kang, 2020), the network will simply move from one drastically incorrect output to the next. The loss and resulting gradient will be large, leading to exploding gradients– the problem that motivates the use of BN in Ardakani et al. (2018).

BN addresses this by normalizing the gate activations. The BN terms are initialized with a mean of 0, and a variance of 1. As the biasing factor γ is not included, the mean remains unchanged. This adjusts the outputs to more moderate values, similar to those in a full-precision LSTM. As such, its function is to enable the network to select optimal binarized parameters without the volatility of large gradients inhibiting convergence.

The use of the batch normalization term BN_c on c_t when computing h_t serves a similar function. c_t is computed using a sum, thereby changing the scale of the values. In computing h_t, renormalizing the values of c_t before computing the product with i_t allows the output to remain reasonably normalized. h_t also serves as the input to subsequent layers. LSTMs, which are inherently sensitive to the distribution of input features (Ba et al., 2016; Hou et al., 2019), therefore benefit from previous layers' output being constrained to a limited scale and normal distribution.

5 Research Question and Hypothesis

The aim of the present work is to examine whether a network trained with binarized LSTMs, per LRBW, could achieve results with a minimal reduction in accuracy, compared with its full-precision counterpart.

Research Question: Is the LRBW method capable of performing ASR? As the authors claim, LRBW is designed to serve as an all-purpose binarization technique for recurrent units. This experiment investigates the degree to which that is the case.

Quantifying the success or failure of such experiments is not trivial. Of course, some minimum accuracy is required in order for an ASR technology to be of any real use. However, how that error is reached is important to the takeaways of the experiment. If the model shows slow, consistent convergence over many training steps, then it may be an indication that a lower error could be achieved with architectural adjustment– an indication of being "on the right track". However, if there is no convergence and the loss over training steps is random noise, that would be cause for a more resounding rejection.

Ardakani et al. (2018) report language modelling results in terms of bits-per-character (BPC) and perplexity, both metrics particular to evaluating language models. Furthermore, these metrics measure cross-entropy, i.e. errors over probability distributions. In keeping with the ASR conventions, the present work compares the word-error-rates. Prior research in binarized ASR reports relative WER increases between 10 and 15% after language model decoding (Xiang et al., 2017; Qian and Xiang, 2019). It is therefore thought sensible to use the upper-bound 15% as a baseline to measure the competitiveness of this method.

Hypothesis: It is hypothesized that an ASR network trained with the LRBW method will produce a result whose WER is constrained to a 15% relative increase, with respect to its full-precision counterpart. This word-error-rate is to be computed after language model decoding.

The notion of applicability entails some nuance. Capturing the "applicability" of LRBW to ASR is therefore tested through 3 experiments:

1. **Basic CTC:** In order to test the sheer applicability of LRBW to ASR, Section 6.2 conducts a basic character-level ASR task with a simple LSTM using the CTC criterion. This is analogous to the single-layer character-level language modelling tasks in Ardakani et al. (2018).

2. **DeepSpeech:** In alignment with the practice of binarizing real, practical ASR architectures (Xiang et al., 2017; Qian and Xiang, 2019), Section 6.3 describes an experiment wherein LRBW is applied to DeepSpeech Hannun et al. (2014), a broadly-used yet notoriously simplistic end-to-end ASR model.

3. **Encoder-Decoder + Attention:** Ardakani et al. (2018) expand the use of LRBW to an attention-equipped encoder-decoder. This is intended to demonstrate the cross-architectural applicability of LRBW. Section 6.4 therefore describes a character-level ASR on an encoder-decoder + attention architecture, as described in Section 3.3.3.

This work can be seen as investigating extensions to the LRBW method, which enable it to be adapted to domains unseen in the original paper. Should the hypothesis be rejected, it would mean that the various deviations from the original method, to be discussed in Section 6.1, were not conducive to the success of LRBW LSTMs on ASR tasks. Whether this means that different deviations ought to be applied, or that the method altogether is inapplicable to ASR, will be a matter of future research.

Regardless acceptance/rejection of the hypothesis, these experiments and their subsequent results are intended to offer better insight into the types of architectures that are best suited to the cross-discipline use of LRBW. Again, as stated in earlier in this section, requiring that a BNN produce a WER within 15% of its full-precision counterpart is somewhat arbitrary. Though models with higher WERs may be rejected, I urge the reader to consider the promise of models with higher WERs which show signs of convergence.

6 Experiments

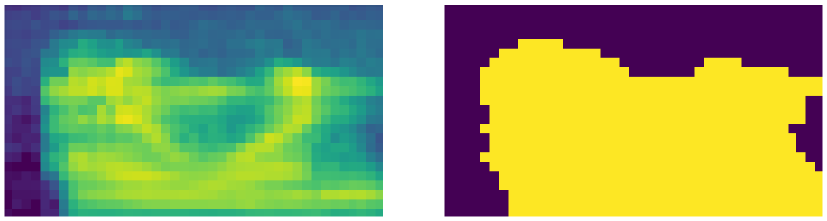
This section describes the experiments performed. All models are trained on the LibriSpeech ASR Corpus (Panayotov et al., 2015) using "train-clean-100" split for training, "dev-clean" for validation, and "test-clean" for final testing. The corpus was obtained through OpenSLR [3][4]. Where applicable, the language model is a 4-gram model built on the LibriSpeech text corpus[5]. The choice of 4-gram as opposed to another value was done out of convenience: a PyTorch wrapper exists with pre-built binaries for a 4-gram language model. The language models were built with KenLM[6].

The models were built in Python with the help of the PyTorch (Paszke et al., 2019) and Torchaudio (Yang et al., 2021) libraries. This includes the wrapper for the language model. Training was performed on the Rijksuniversiteit Groningen's high-performance computing (HPC) cluster, *Peregrine*. Training was done on 2 nodes of a single NVIDIA® V100 Tensor Core[7] GPU with 32GB of GPU memory.

6.1 Deviations from the Original LRBW Experiments

In the spirit of minimizing variables, the first experiments are designed to, as closely as possible, replicate the language modelling experiments in Ardakani et al. (2018). A

[3]`https://www.openslr.org/`
[4]the LibriSpeech ASR corpus is found at `https://www.openslr.org/12`
[5]`http://www.openslr.org/11`
[6]`https://kheafield.com/code/kenlm/`
[7]`https://www.nvidia.com/en-us/data-center/v100/`



(a) full-precision (b) binarized

Figure 5: Full-precision *(a)* and binarized *(b)* log-mel spectrograms of the beginning of the utterance "I wonder".

number of notable modifications are made in order to adapt the experiment to the speech domain, and, more specifically, to ASR. Furthermore, some additional, arguably gratuitous tweakings have been made in order to improve the performance of the networks, with the hopes of motivating real-life utility of said networks.

6.1.1 Input Features

Ardakani et al. (2018) perform NLP experiments, taking as input features either characters or words. Their inputs are therefore one-hot encoded vectors. Per the explanation in 2.2.2, the matrix multiplications between input/hidden features $x|h$ and weights W is a simple matter of indexing a row of W. In the case of ASR, where inputs are speech features, i.e. k-dimensional vectors of floating-point values, some binarized or equally "simple" (i.e. one-hot vector) inputs must be provided. If real-values speech features are passed as inputs, the binary matrix multiplication described in 2.2.2 is not possible, thereby negating the benefits of the compression that binarization affords, allthewhile suffering from the imprecision. The options for providing "simple" inputs are as follows:

- Binarize the speech features
- Begin with full-precision layer

Binarizing speech features

The first option is to extract speech features, in this case log-mel filterbanks, and to constrain the values to ± 1. In order to illustrate the effects of this, 5 demonstrates two representations of the same spectrogram: the first (a) is the original, and the second (b) is a binarized version of (a). The spectrogram is composed of 23 log-mel filterbanks using the *torchaudio* (Yang et al., 2021) module `torchaudio.complicance.kaldi`'s `fbank`[8] function. The spectrogram was binarized by normalizing the original spectrogram and performing deterministic binarization as shown in 1.

As can be seen in 5, far too much information is lost in binarizing speech features. In this case, the segment shown is the beginning of a male speaker uttering "I wonder",

[8]`https://pytorch.org/audio/stable/compliance.kaldi.html`

so, "I won–". Transcribed in the International Phonetic Alphabet (IPA), this would be [a͡ɪ.ˈwʌn]. The formant movement, visible in (a), indicates the articulated phonemes. In binarization, the distinction between formants is entirely lost, indicating only that there is energy within a certain frequency range. Binarizing speech features is therefore not an option.

Beginning with a full-precision layer

Drawing inpiration from previous research in binarized ASR (Xiang et al., 2017; Qian and Xiang, 2019), this option first passes the speech features through a full-precision layer, before binarizing the layer's outputs/subsequent layer's inputs. Whether a fully-connect or RNN layer should be used as the first is to be explored. A fully-connected layer would save on parameters and has the advantage of being parallelizable. The savings on computation are advantageous, as BNNs are remarkably slower to train (Courbariaux et al., 2015; Ardakani et al., 2018). However, there is the possiblity that temporal dependencies preserved across an LSTM forward pass may be crucial to overcome the loss of precision introduced in binarization. See 6.2 for a discussion thereupon.

Adding an extra layer is a significant deviation from Ardakani et al. (2018), whose experiments use one recurrent layer. However, the granularity of information of speech features is remarkably greater than that of one-hot encoded characters. In order to preserve this information, extra numerical precision is a must.

6.1.2 Bidirectionality

The experiments in this work make use of bidirectional LSTMs. The basic language modelling experiments in Ardakani et al. (2018), on the other hand, use single-direction networks. The reasoning for this is simple: Ardakani et al. (2018) take as input a sequence of characters or words and train the network to predict the following character/word. It goes without saying that knowledge of future inputs in a task that predicts future inputs renders the task useless. For obvious reasons, this is not the case with ASR.

The experiments in this work compare the difference in performance between equivalent full-precision and binarized networks. Therefore, no absolute baseline performance is required. For this reason, the performance improvements afforded by bidirectionality may be seen as gratuitous for the purpose of these experiments. However, it is the hope that this work will inspire real-life use-cases for BNNs. In practice, bidirectionality is commonplace in RNN-based ASR, it improves the performance, is easily implemented, and is not "cheating", as it would be with next-token language modelling. Since a reasonable absolute performance will inspire confidence that such methods have applicability in industry, this work includes bidirectional LSTMs.

6.2 Experiment 1: Simple CTC

In order to test the general applicability of LRBW LSTM in its most basic form to ASR, a character-level speech recognition task was conducted. For this experiment, 23 log-mel filterbanks were extracted from LibriSpeech recordings with a window size of 25ms and a stride of 10ms. Inputs were fed either directly to a 2-layer LSTM or first through a feed-forward network, consisting of a fully-connected layer and a ReLU activation clipped at 20, per Hannun et al. (2014) before being fed to an LSTM. In the case of binary training, these outputs were renormalized with a mean of 0 and standard deviation of 1 in order to avoid having the binarized inputs to the binarized LSTM be nearly– of not all 1s. The outputs of the LSTMs were fed through a fully-connected layer in order to have the output size match the the number of characters. A log-softmax was computed over the outputs.

All hidden layers had 512 units. The batch size was 64, using an Adam optimizer (Kingma and Ba, 2014) and a starting learning rate of $1e^{-3}$. Biases were not used on LSTM layers, as the LRBW method does not account for biases. Validation was performed after every 100 training steps, and training ran for 24 or 72 hours, depending on the task.

The variables tested were: binarized vs. full-precision LSTM, bidirectional vs single-direction LSTM, and linear-projected input vs straight-to-LSTM input. Note that, per Ardakani et al. (2018), only the LSTMs were binarized.

6.3 Experiment 2: DeepSpeech

In order to test the feasibility of LRBW on real-world applications, as is the case with earlier BNN-based ASR research (Xiang et al., 2017; Qian and Xiang, 2019; Gao et al., 2021), an architecture similar to Baidu's DeepSpeech (Hannun et al., 2014), a popular and reasonably simple CTC-based acoustic model was tested. The model consists of 3 feed-forward layers, a bidirectional "vanilla" RNN, and a linear output. My implementation has 4 notable deviations:

1. In the original model, the network input x_t is the set of speech features at time t as well as the speech features for 5, 7, or 9 time steps before and after t. For the sake of simplicity and memory conservation, this experiment used only the features at time t
2. The input features in the original DeepSpeech model are linearly-spaced, non-mel spectral features, computed with a window size of 20ms (and a stride of 10ms). Again, for simplicity, the same features as in 6.2, i.e. 23 log-mel filterbanks, were used. This was done in order to minimize variables between the different experiments.
3. The recurrent unit, per the LRBW method, is an LSTM, instead of a vanilla RNN.
4. Per LRBW, again, no biases were used.

All hidden layers had 2048 units, and a dropout of 0.1 was applied to the first 3 fully-connected layers. The same optimization parameters were used as in 6.2. Training was allowed to continue for 72 hours, saving the model with the lowest validation CER.

6.4 Experiment 3: Encoder-Decoder with Attention

The final experiment in Ardakani et al. (2018) involves a question-answering task with a model replicating Deepmind's Attentive Reader (Hermann et al., 2015). Given the successful application of LRBW to such a task, the present work applies the LRBW method to an encoder-decoder + attention ASR model. This experiment attempts to further corroborate the authors' claim that LRBW can extend to diverse sequence-to-sequence tasks. The model architecture for this experiment is inspired primarily by Bahdanau et al. (2016). This architecture was chosen for the fact that it represents the first character-level experiment using the encoder-decoder + attention architecture trained on non-proprietary data (as opposed to Google's Listen, Attend, and Spell Chan et al. (2015)). Details about the architecture follow below.

The input features are 40 log-mel filterbanks and their energies computed with a window size of 25ms and a stride of 10ms. Their deltas and delta-deltas were computed, for a total of 123 input features. The encoder is a 4-layer bidirectional LSTM with 256 hidden units per layer, where, after the 3rd and 4th layer, every second output was dropped, for a total downsampling factor of 4 (i.e. 2×2). In other words, the encoder output to which the attention mechanism "attends" is a quarter the length of the input.

The attention mechanism uses learned location awareness, meaning that learned parameters are used in order to encourage monotonic alignment and reduce the search space at each decoder step, as explained in 3.3.3. This is done with a 1-dimensional convolution with a kernel size of 201, representing a center frame and 100 frames on either side. All inputs to the attention mechanism, i.e. the decoder hidden state and the encoder outputs, were passed through a linear layers with an output size of 256. Distinct *query*, *key*, and *value* projections, per Vaswani et al. (2017) are used, all of dimension 256. The decoder was a single-layer LSTM with a hidden size of 512, and a linear layer projecting the outputs to the number of characters.

Adjusting for memory constraints, batch sizes of 16 for full-precision and 8 for binarized training were used. A baseline full-precision and binarized network were trained. For the latter, only the LSTM layers were binarized, meaning the attention mechanism and output linear projections in the binarized network was full-precision. This is the same as in Ardakani et al. (2018). An Adadelta (Zeiler, 2012) optimizer with a starting learning rate of 1 and $\rho = 0.95$ and $\epsilon = 1e^{-8}$ was used. Training was allowed to run for 72 hours, with validation every 1000 steps. The model with the lowest validation loss was kept for testing.

This experiment used a fork of an existing repository (Liu et al., 2019), originally built as an end-to-end speech recognition toolkit. The fork extends this repository to include BNNs.

7 Results & Discussion

This section presents and analyzes the results of the experiments discussed in Section 6.

7.1 Experiment 1: Simple CTC

The first experiment tests the viability of adapting LRBW to basic 1- or 2-layer LSTMs. The performance of their full-precision baselines (Section 7.1.1) and binarized counterparts (Section 7.1.2 are compared and discussed here.

7.1.1 Training

Let us first begin with the training metrics. During training, the greedy-decoded character error rates (CER) were computed on the validation set after every 100 steps. The architectures and their respective minimum CERs are recorded in Table 2.

	full-precision		bnn	
	CER	step	CER	step
2-layer	0.1916	40.8K	0.7366	21.3K
2-layer (bidir)	**0.1238**	29.6K	0.5930	5.0K
linear proj	0.2647	41.3K	0.5094	6.9K
linear proj (bidir)	0.2385	13.9K	**0.3918**	7.1K

Table 2: Minimum CER calculations on the LibriSpeech "dev-clean" split on full-precision and binarized ("bnn") networks. "2 layer" refers to a 2-layer LSTM network. "linear proj" refers to a linear layer feeding into a single LSTM layer. "bidir" indicates the use of bidirectional LSTMs. "CER" refers to the minimum CER, and "step" refers to the training step after which this CER was achieved.

As one can see from Table 2, the added LSTM, rather than simple linear layer improved training for full-precision netowrks. Curiously, the opposite is true of BNNs. In all cases bidirectionality improved the networks' performances. It should be noted that bidirectionality and the use of LSTMs rather than linear layers come at the cost of additional parameters. Not accounting for biases, swapping a fully-connected layer for an LSTM multiplies the number of weights 8-fold, and bidirectionality doubles the number of weights in an LSTM.

7.1.2 Comparing the full-precision and binarized during testing

For this comparison, the networks were used to compute outputs on the LibriSpeech "test-clean" split. The outputs were decoded using beam search decoding on a 4-gram language model built with the LibriSpeech text corpus, using a beam width of 500 and language model weight of 5.0. The WER was then computed on the beam search outputs.

	fp	bin
2-layer	0.3084	0.9580
2-layer (bidir)	**0.1576**	0.9831
linear proj	0.6636	0.8777
linear proj (bidir)	0.3209	**0.7152**

Table 3: WER calculations on various architectures. "2 layer" refers to a 2-layer LSTM network. "Linear proj" refers to a linear layer feeding into a single LSTM layer. "Bidir" indicates the use of bidirectional LSTMs. "fp" indicates that the LSTMs used full-precision weights, whereas "bin" means the weights are binary, trained with LRBW.

Due to the increased resource demand, BNN training underwent considerably fewer steps than full-precision networks. Depending on the number of LSTM layers and bidirectionality, BNNs trained as much as 2.7 $\times$ slower than their respective full-precision counterparts.

7.2 Experiment 2: DeepSpeech

The full precision network achieved a minimum CER of 0.2827 after 34.1K training steps. This is considerably higher than the best simple CTC results, and higher still than the worst results. No specific reports of CER on the validation data is reported in the original DeepSpeech publication (Hannun et al., 2014). However, these results are remarkably worse than one would expect. A few reasons could include the use of single time step feature vectors rather than context windows, the absence of biases, the smaller hidden size (1024 rather than 2048), the much smaller datasets used in training, and the smaller number of input features (23 rather than 80). These changes were mostly made in compliance with the availability of training hardware.

Although such troubleshooting is not necessarily directly pertinent to the present research, it is the hope that the reader can use this to better understand the vability of binarization on certain architectures. Given that training BNNs is resource-intensive, the choice of where to add storage complexity, in terms of input features, biases, or hidden units, will be a trade-off that the reader must make in considering their own hardware limitations.

Figure 6 Shows the greedy-decoded CER on the validation set of the binarized DeepSpeech model after every 100 training steps. Note a rising CER to around 6.5, before settling on a CER of 1.0, i.e. 100% at step 7.8K, where it remained until training stopped. This represents the model learning to predict empty strings, whereas the high CERs coincide with long strings of random repeated characters. As there is no convergence, no WERs on the test set were conducted.

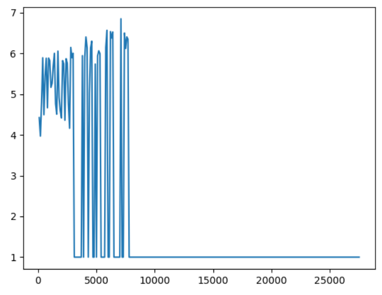


Figure 6: CER by training step of the binarized DeepSpeech model.

7.3 Experiment 3: Encoder-Decoder + Attention

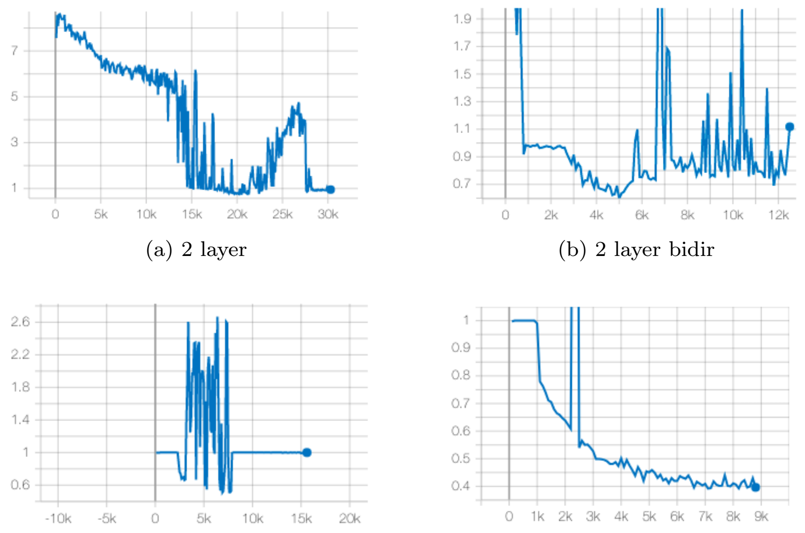
The baseline network exhibited convergence until training was attenuated (again, resource constraints), achieving a minimum validation loss of 0.4202 at step 44.6K. Nonetheless, the final CER on the validation set at the end of training was 0.1794, nearly 18%. Despite these promising results, the binarized network failed to achieve a CER below 1. This is, by far, the most resource-demanding network, thereby only undergoing 7.2K training steps. Again, testing such a network would not be of any use.

7.4 Discussion

Given that no BNNs were trained with performances anywhere near the results of their full-precision counterparts, the hypothesis is rejected. Nonetheless, a number of takeaways can be used to inspire future successes. They are discussed here.

Figure 7 plots the CERs of the BNNs trained in Experiment 1 (Section 6.2. It is apparent that, of all the networks, only (d), namely the bidirectional linear projected network, shows convergence. This ignores the spike around step 2,000. Although it is difficult to see from the graph, there is a consistent increase in CER starting around step 7,000, signalling the end of convergence and the beginning of overfitting. While this is the clearest picture of convergence, there is potentially a case to be made for (b), i.e. the bidirectional 2-layer LSTM, up until step 5,000. Starting around step 2,600, there is a consistent decrease in CER, which may signal the potential for this network to better converge with the proper tweakings. Note that the other two networks appear more-or-less hopeless, with an affinity for CERs far greater than 100% (a) and random noise punctuated by dead predictions, signalling empty predictions (b).

It appears that bidirectional LSTMs are the best bet for constructing LRBW ASR networks. Networks built thereupon have either achieved the greatest CERs in binary training or hinted at convergence. This offers us insight into the importance of the temporal dependencies in performing ASR. It may be tempting to select the best-performing BNN in experiment 1, namely the bidirectional linear-projected network, as the only



(a) 2 layer (b) 2 layer bidir

(c) linear proj (d) linear proj bidir

Figure 7: Plots of the valid CERs over time-steps for the BNNs from Experiment 1.

promising contender. I would nonetheless caution the reader against discounting the less performant BNN architectures. 39% is still a very high CER, and improvements that benefit this network may very well benefit other architectures as well.

It appears that binarization has taken better to the linear projected networks in Experiment 1 (6.2) than 2-layer LSTMs, despite the opposite being true of full-precision networks. One would expect the LSTM's greater number of parameters to be preferable, especially given the BNN's poverty of precision. This could have to do with the difference in activation functions. Linear outputs are subject to the ReLU activation and then renormalized to a mean of 0 and standard deviation of 1 before being binarized. This normalization could possibly be superior to the LSTM output for the purposes of binarization. The strategic use of normalization is suggested in Section 9.2 as a potential approach to future research.

Given the superior performance of binarized LSTMs atop fully-connected layers, it is surprising that the DeepSpeech model in Experiment 2 (Section 6.3, being remarkably similar except for having a greater number of fully-connected layers and hidden units, took so poorly to binarization. This may speak to a diminishing returns phenomenon, whereby increasing the number of layers and/or input units ceases to improve binarized LSTMs beyond a certain point, and begins to hamper their performance.

Figure 8 is a plot of the training and validation losses of the binarized attention network from Experiment 3 (Section 6.4). Despite the losses being very large values (minimum of 2.851 for validation), this curve could potentially suggest convergence. Alternatively, 7K steps may simply be too few to have a proper picture of the training. This will have to be examined in research with resources that permit as much.

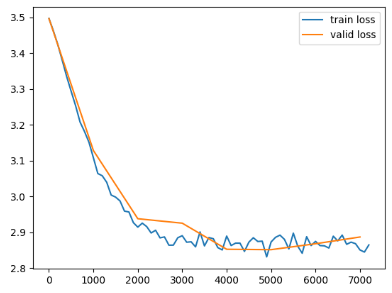


Figure 8: Training (blue) and validation (orange) losses of the binarized encoder-decoder+attention model. Note that the smoother validation loss is due to the validation loss' only being evaluated at every 1000 steps, as opposed to every 100 for the training loss.

It is possible that the inputs are too long and numerically dense for LRBW. The authors conduct experiments with inputs of up to 1000 time steps, with a slight decrease in performance as the number of steps increase. The input sequences in my experiments, i.e. audio features, can be considerably longer. Additionally, these experiments investigate character-level language modelling on one-hot encoded vectors. This means that the inputs are constrained to N_c possible values, N_c denoting the number of possible characters. The sample space for speech features is considerably more dense, given that each feature in a vector of N_f features can be one of 2^P possible values, where P is the numerical precision with which these features are represented.

The depth of ASR networks may just as well be a source of performance degradation. The original implementation investigates the method on relatively shallow networks. In the interest of applying this method to ASR, as has been discussed in Section 6.1, deeper networks were used.

It stands to reason, also, that the viability of binarizing a given architecture relies on some performance threshold. Architectures with lower baseline performances tended not to demonstrate any convergence in binarized training.

The first and most obvious potential avenue to pursue would be adding more hidden units. Many of the experiments in Ardakani et al. (2018) use networks with 1024 hidden units, compared to the 512 in experiment 1 (Section 6.2). This was done in order to cooperate with resource contraints, as well as the double-sized outputs of bidirectional LSTMs, which Ardakani et al. (2018) does not utilize.

8 Conclusion

This work is a first attempt at the use of binarized LSTMs for end-to-end automatic speech recognition (ASR). This has been done using the LRBW (Ardakani et al., 2018) method, and adapted it to ASR with the inspiration of earlier attempts to apply BNNs to ASR (Xiang et al., 2017; Qian and Xiang, 2019; Gao et al., 2021). It was the goal of this research to produce a binarized network whose performance degradation with respect to its full-precision counterpart was constrained to a relative WER increase of 15%. Although this was not achieved, the experiments have made a number of contributions to research in BNNs as well as in ASR. These contributions include:

- The use of BNNs has been tested for the first time on end-to-end ASR.
- A greater insight into the importance of temporal dependencies in recurrent ASR has been achieved.
- The importance of numerical precision in representing speech features has been better demonstrated.
- There now exists an open-source, modular, easy-to-use codebase for building and training LRBW-based networks.

9 Future Work

Subsequent researchers now have a basis atop which they should seek to find a successful LRBW ASR network. Such future research should experiment with the variables and hyperparameters used in this research. These include:

- Finding the minimum number of parameters that can capture the information in an ASR BNN. In other words, a hidden size of 512 may be too small. Doubling this to 1024 would still mean massive storage reductions, and would allow for much greater detail, despite the binarization. This could also serve as a compromise between the sizes of the fully-connected networks in Experiment 1 and the DeepSpeech-based network in Experiment 2, as the size of the latter is speculated to be a hindrance to the model's performance.
- Using smaller datasets in the hopes of contraining the feature space such that it can be captured with BNNs.
- Experimenting with normalization parameters, both within the binarized LSTM cell, and when binarizing the data prior to passing it to the cell.

The following sections suggest broader avenues of future research.

9.1 Faster BNNs

The sheer amount of time it takes to train BNNs is a hindrance to the advancement of research in the field. Improving the training speed of BNNs involves low-level parallelization of the LSTM's constituents. This entails a knowledge of C++ and computing hardware that was quite simply beyond the scope of this research.

9.2 Normalization

As mentioned, LSTMs and their internal gates are notoriously sensitive to the distributions of their inputs. Future work should seek to perform statistical evaluations that consider the effects of normalization on training binarized LSTMs. Ardakani et al. (2018) uses this tactic in developing the LRBW method, and future experiments could benefit from extending this to ASR-specific problems, such as different speech features, their respective distributions, the effects of normalizing them, etc.

Furthermore, per the use of batch normalization within the LRBW LSTM cell, future work could investigate whether binarized LSTMs beenfit from batch normalization on their binarized inputs, counter to the conventional wisdom (Ba et al., 2016).

9.3 Knowledge Distillation

Distilling knowledge from a large network to a smaller one is a tactic used frequently in network compression. As discussed earlier, it has proven successful in binarizing networks for hybrid ASR (Qian and Xiang, 2019). Given the results of the basic CTC experiments discussed in Section 7, there is reason to believe that Knowledge distillation could afford improvements. To understand this, we now consider the technicalities of knowledge distillation.

Knowledge distillation uses the predictions from a larger, *teacher* network as labels in training the smaller/compressed/quantized *student* network. In the case of Qian and Xiang (2019), the teacher is a full-precision network and the student is its binarized counterpart. This is helpful in that the student network learns not only the correct labels, but also the interconnection between similar labels.

Consider the output of a network $\hat{y}$ in a multi-class classification task. For the sake of simplicity, let us consider that the probabilities are between 0 and 1 and sum to 1, and not in log space. This distribution represents the probabilities of a given class. Represented as a vector, these would be:

$$\hat{y} = \{0.04, 0.01, \ldots, 0.12\}$$

The label y would be a one-hot vector, the value 1 being assigned to the index of the correct class. The loss function would therefore encourage parameters that maximize the output at the correct index and minimize them at the incorrect one.

However, if the label were itself a probability distribution from a more precise, more robust, and better-trained network, then the loss function would not only encourage the greatest probability at the correct index, but an output distribution that best represents the given input. This has a number of advantages. For one, it distinguished between "obvious" and "ambiguous" outputs. The student network can learn when an input unambiguously belongs to a certain class, and when the label of the training example is less clear. In other words, the student network is encouraged to seek which input feature values are characteristic of which classes. This nuance is taken for granted in large networks, but the ability to learn fine features is not necessarily a given in smaller ones.

The other advantage is contextual awareness. The teacher network, having learned features over many training steps across a variety of contexts, can predict outputs based not only on input features, but surrounding inputs and/or outputs. In the case of CTC, this would be the outputs $\hat{y}_t$ reflecting the possible characters at a given time-step t, taking into account acoustic features and their hidden representations at timesteps other than t. This is especially nuanced in CTC-based ASR, where the predictions at t do not take into account previous/future predictions. The teacher network has therefore learned to predict the character based on the character predictions that "may" be made at timesteps surrounding t. Again, while such detail may be available to larger networks, that may very well not be the case with smaller ones.

9.4 Attention

To my knowledge, no work exists on an attention-specific binarization technique. Should binarization eventually work on the encoder-decoder + attention model, the greatest bottleneck, in terms of storage and computational power, would be the attention module. This invites the opportunity to further shrink the network with binarized attention modules.

Furthermore, transformer-based architectures (Vaswani et al., 2017) have come to occupy much of the ASR landscape with models such as Wav2Vec 2.0 (Baevski et al., 2020) and Conformer (Gulati et al., 2020). Transformers, built upon self-attention, an extension to the attention module discussed in Section 3.3.3, could potentially see massive reductions in terms of their computational footprint with the advent of binarized attention modules.

References

Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Cheng, Q., Chen, G., Chen, J., Chen, J., Chen, Z., Chrzanowski, M., Coates, A., Diamos, G., Ding, K., Du, N., Elsen, E., Engel, J., Fang, W., Fan, L., Fougner, C., Gao, L., Gong, C., Hannun, A. N., Han, T., Johannes, L. V., Jiang, B., Ju, C., Jun, B., Legresley, P., Lin, L., Liu, J., Liu, Y., Li, W., Li, X., Ma, D., Narang, S., Ng, A., Ozair, S., Peng, Y., Prenger, R., Qian, S., Quan, Z., Raiman, J., Rao, V., Satheesh, S., Seetapun, D., Sengupta, S., Srinet, K., Sriram, A., Tang, H., Tang, L., Wang, C., Wang, J., Wang, K., Wang, Y., Wang, Z., Wang, Z., Wu, S., Wei, L., Xiao, B., Xie, W., Xie, Y., Yogatama, D., Yuan, B., Zhan, J., and Zhu, Z. (2015). Deep speech 2: End-to-end speech recognition in english and mandarin. *33rd International Conference on Machine Learning, ICML 2016*, 1:312–321.

Ardakani, A., Ji, Z., Smithson, S. C., Meyer, B. H., and Gross, W. J. (2018). Learning recurrent binary/ternary weights. *7th International Conference on Learning Representations, ICLR 2019*.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450.*

Baevski, A., Zhou, Y., Mohamed, A., and Auli, M. (2020). wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in Neural Information Processing Systems*, 33:12449–12460.

Bahdanau, D., Cho, K. H., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings.*

Bahdanau, D., Chorowski, J., Serdyuk, D., Brakel, P., and Bengio, Y. (2016). End-to-end attention-based large vocabulary speech recognition. *IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 4945–4949.

Bello, I., Zoph, B., Vaswani, A., Shlens, J., and Le, Q. V. (2019). Attention augmented convolutional networks. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3286–3295.

Bengio, Y., Léonard, N., and Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432.*

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5:157–166.

Biewald, L. (2019). Deep learning and carbon emissions. Towards Data Science.

Chan, W., Jaitly, N., Le, Q. V., and Brain, V. G. (2015). Listen, attend and spell. *arXiv preprint arXiv:1508.01211.*

Chiu, C. C., Sainath, T. N., Wu, Y., Prabhavalkar, R., Nguyen, P., Chen, Z., Kannan, A., Weiss, R. J., Rao, K., Gonina, E., Jaitly, N., Li, B., Chorowski, J., and Bacchiani, M. (2018). State-of-the-art speech recognition with sequence-to-sequence models. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2018-April:4774–4778.

Cho, K., van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *Proceedings of SSST 2014 - 8th Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111.

Chorowski, J., Bahdanau, D., Cho, K., and Bengio, Y. (2014). End-to-end continuous speech recognition using attention-based recurrent nn: First results. *arXiv preprint arXiv:1412.1602.*

Chorowski, J., Bahdanau, D., Serdyuk, D., Cho, K., and Bengio, Y. (2015). Attention-based models for speech recognition.

Cooijmans, T., Ballas, N., Laurent, C., Çağlar Gülçehre, and Courville, A. (2016). Recurrent batch normalization. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings.*

Courbariaux, M., Bengio, Y., and David, J.-P. (2015). Binaryconnect: Training deep neural networks with binary weights during propagations.

Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y., and Com, Y. U. (2016). Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. *arXiv preprint arXiv:1602.02830*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

Dong, L., Xu, S., and Xu, B. (2018). Speech-transformer: A no-recurrence sequence-to-sequence model for speech recognition. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2018-April:5884–5888.

Edel, M. and Köppe, E. (2016). Binarized-blstm-rnn based human activity recognition. *2016 International Conference on Indoor Positioning and Indoor Navigation, IPIN 2016*.

Gao, S., Wang, R., Jiang, L., and Zhang, B. (2021). 1-bit wavenet: Compressing a generative neural network in speech recognition with two binarized methods. *Proceedings of the 16th IEEE Conference on Industrial Electronics and Applications, ICIEA 2021*, pages 2043–2047. Need IEEE access to get ahold of this one.

Gardner, E. and Derrida, B. (1988). Optimal storage properties of neural network models. *Journal of Physics A: Mathematical and General*, 21:271.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR.

Golea, M. and Marchand, M. (1993). On learning perceptrons with binary weights. *Neural Computation*, 5(5):767–782.

Graves, A. (2006). Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. *ACM International Conference Proceeding Series*, 148:369–376.

Graves, A. and Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. pages 1764–1772.

Graves, A., Mohamed, A. R., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pages 6645–6649.

Gulati, A., Qin, J., Chiu, C. C., Parmar, N., Zhang, Y., Yu, J., Han, W., Wang, S., Zhang, Z., Wu, Y., and Pang, R. (2020). Conformer: Convolution-augmented transformer for speech recognition. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 2020-October:5036–5040.

Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., and Ng, A. Y. (2014). Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*.

He, Y., Sainath, T. N., Prabhavalkar, R., McGraw, I., Alvarez, R., Zhao, D., Rybach, D., Kannan, A., Wu, Y., Pang, R., Liang, Q., Bhatia, D., Shangguan, Y., Li, B., Pundak, G., Sim, K. C., Bagby, T., Chang, S. Y., Rao, K., and Gruenstein, A. (2018). Streaming end-to-end speech recognition for mobile devices. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2019-May:6381–6385.

Hermann, K. M., Kočiský, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. (2015). Teaching machines to read and comprehend. *Advances in Neural Information Processing Systems*, 2015-January:1693–1701.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9:1735–1780.

Hou, L., Yao, Q., and Kwok, J. T. (2016). Loss-aware binarization of deep networks. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*.

Hou, L., Zhu, J., Kwok, J., Gao, F., Qin, T., and Liu, T.-y. (2019). Normalization helps training of quantized lstm. *Advances in Neural Information Processing Systems*, 32.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. (2016). Binarized neural networks. *Advances in neural information processing systems*, 29.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *32nd International Conference on Machine Learning, ICML 2015*, 1:448–456.

Kang, M.-J. (2020). Comparison of gradient descent for deep learning. *Journal of the Korea Academia-Industrial cooperation Society*, 21(2):189–194.

Kim, J., Hwang, K., and Sung, W. (2014). X1000 real-time phoneme recognition vlsi using feed-forward deep neural networks. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pages 7510–7514.

Kim, S., Hori, T., and Watanabe, S. (2017). Joint ctc-attention based end-to-end speech recognition using multi-task learning. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pages 4835–4839.

Kingma, D. P. and Ba, J. L. (2014). Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings.*

Kukačka, J., Golkov, V., and Cremers, D. (2017). Regularization for deep learning: A taxonomy. *arXiv preprint arXiv:1710.10686.*

Köhler, H., Diederich, S., Kinzel, W., and Opper, M. (1990). Learning algorithm for a neural network with binary synapses. *Zeitschrift für Physik B Condensed Matter 1990*, 78:333–342.

Liu, A. H., Sung, T.-W., Chuang, S.-P., yi Lee, H., and shan Lee, L. (2019). Sequence-to-sequence automatic speech recognition with word embedding regularization and fused decoding.

Liu, X., Cao, D., and Yu, K. (2018). Binarized lstm language model. *NAACL HLT 2018 - 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, 1:2113–2121.

Lu, L., Zhang, X., Cho, K., and Renals, S. (2015). A study of the recurrent neural network encoder-decoder for large vocabulary speech recognition. In *Sixteenth Annual Conference of the International Speech Communication Association.*

Mishra, R., Gupta, H. P., and Dutta, T. (2020). A survey on deep neural network compression: Challenges, overview, and solutions. *arXiv preprint arXiv:2010.03954.*

Ott, J., Lin, Z., Zhang, Y., Liu, S.-C., and Bengio, Y. (2016). Recurrent neural networks with limited numerical precision. *arXiv preprint arXiv:1608.06902.*

Palaz, D., Collobert, R., and Magimai-Doss, M. (2013). Estimating phoneme class conditional probabilities from raw speech signal using convolutional neural networks. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, pages 1766–1770.

Panayotov, V., Chen, G., Povey, D., and Khudanpur, S. (2015). Librispeech: An asr corpus based on public domain audio books. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2015-August:5206–5210.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Qian, Y. M. and Xiang, X. (2019). Binary neural networks for speech recognition. *Frontiers of Information Technology Electronic Engineering 2019 20:5*, 20:701–715.

Raffel, C., Luong, M.-T., Liu, P. J., Weiss, R. J., and Eck, D. (2017). Online and linear-time attention by enforcing monotonic alignments. *Proceedings of the 34th International Conference on Machine Learning*, pages 2837–2846.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. (2016). Xnor-net: Imagenet classification using binary convolutional neural networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9908 LNCS:525–542.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, pages 533–536.

Rupali, M., Chavan, S., and Sable, G. S. (2013). An overview of speech recognition using hmm international journal of computer science and mobile computing an overview of speech recognition using hmm. *IJCSMC*, 2:233–238.

Sainath, T. N., Kingsbury, B., Mohamed, A. R., and Ramabhadran, B. (2013). Learning filter banks within a deep neural network framework. *2013 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2013 - Proceedings*, pages 297–302.

Scheidl, H., Fiel, S., and Sablatnig, R. (2018). Word beam search: A connectionist temporal classification decoding algorithm. *Proceedings of International Conference on Frontiers in Handwriting Recognition, ICFHR*, 2018-August:253–258.

Schneider, S., Baevski, A., Collobert, R., and Auli, M. (2019). wav2vec: Unsupervised pre-training for speech recognition. *arXiv preprint arXiv:1904.05862*.

Simons, T. and Lee, D. J. (2019). A review of binarized neural networks. *Electronics 2019, Vol. 8, Page 661*, 8:661.

Solla, S. A. and Winther, O. (1998). Optimal perceptron learning: an online bayesian approach.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 4:3104–3112.

Trehan, D. (2020). Gradient descent explained. Towards Data Science.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Łukasz Kaiser, and Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 2017-December:5999–6009.

Watanabe, S., Member, S., Hori, T., Kim, S., Member, S., Hershey, J. R., and Hayashi, T. (2017). Hybrid ctc/attention architecture for end-to-end speech recognition. *IEEE Journal of Selected Topics in Signal Processing*, 11:1240–1253.

Xiang, X., Qian, Y., and Yu, K. (2017). Binary deep neural networks for speech recognition.

Xu, C., Yao, J., Lin, Z., Ou, W., Cao, Y., Wang, Z., and Zha, H. (2018). Alternating multi-bit quantization for recurrent neural networks. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*.

Yang, Y.-Y., Hira, M., Ni, Z., Chourdia, A., Astafurov, A., Chen, C., Yeh, C.-F., Puhrsch, C., Pollack, D., Genzel, D., Greenberg, D., Yang, E. Z., Lian, J., Mahadeokar, J., Hwang, J., Chen, J., Goldsborough, P., Roy, P., Narenthiran, S., Watanabe, S., Chintala, S., Quenneville-Bélair, V., and Shi, Y. (2021). Torchaudio: Building blocks for audio and speech processing. *arXiv preprint arXiv:2110.15018*.

Yuan, C. and Agaian, S. S. (2021). A comprehensive review of binary neural network. *arXiv preprint arXiv:2110.06804*.

Zeiler, M. D. (2012). Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Zeyer, A., Irie, K., Schlüter, R., and Ney, H. (2018). Improved training of end-to-end attention models for speech recognition. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, pages 7–11.

Zhang, Y., Chan, W., and Jaitly, N. (2017). Very deep convolutional networks for end-to-end speech recognition. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pages 4845–4849.